GERAÇÃO AUTOMÁTICA DE CÓDIGO PARA EXECUÇÃO EM UM AMBIENTE DE COMPUTAÇÃO EM DATAFLOW

LEONARDO LUIZ PADOVANI DA MATA

GERAÇÃO AUTOMÁTICA DE CÓDIGO PARA EXECUÇÃO EM UM AMBIENTE DE COMPUTAÇÃO EM DATAFLOW

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais — Departamento de Ciência da Computação como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Renato Antônio Celso Ferreira
Co-orientador: Fernando Magno Quintão Pereira

Belo Horizonte, Minas Gerais

Agosto de 2010

da Mata, Leonardo Luiz Padovani
M425g Geração automática de código para execução em um ambiente de computação em DataFlow / Leonardo Luiz Padovani da Mata. — Belo Horizonte, Minas Gerais, 2010
xxv, 239 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais — Departamento de Ciência da Computação
Orientador: Renato Antônio Celso Ferreira
Co-orientador: Fernando Magno Quintão Pereira

1. Computação — Teses.
2. Arquitetura de Computador — Teses.
3. Programação de Sistemas (computação) — Teses.
4. Compiladores (Programas de Computador) — Teses
I. Orientador. II. Coorientador. III. Título.

CDU 519.6*21(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Geração automática de código para a execução em um ambiente de computação em DataFlow

LEONARDO LUIZ PADOVANI DA MATA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. RENATO ANTÔNIO CELSO FERREIRA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. FERNANDO MAGNO QUINTÃO PEREIRA - Co-orientador
Departamento de Ciência da Computação - UFMG

PROF. MÁRCIO LUIZ BUNTE DE CARVALHO
Departamento de Ciência da Computação - UFMG

PROF. WAGNER MEIRA JUNIOR
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 18 de outubro de 2010.

Dedico essa dissertação à memória do professor Christiano Gonçalves Becker

Agradecimentos

Agradeço ao meu Pai e minha Mãe por me apoiarem em todas as decisões da minha vida, suportarem todas aquelas que não concordavam e me aconselharem em todas aquelas que aviam resalvas. Os exemplos deles me moldaram como pessoa. Agradeço minha avó Inísia por mostrar que o caminho da bondade e da gentileza é o caminho que mais engrandece uma pessoa. A minha família, primos, tios, deixo um grande beijo. Obrigado pela força vinda de vocês.

A Lara, meu amor, que suportou todos as dificuldades comigo, e me ajudou a reencontrar o caminho correto. Te amo Ti!

Aos meus amigos, Leonel, Ferreira, Leandro, Rocha, Tamás, Chuchú, Rerussar, Macambira, Fireman, Rodrigo, Débora, Cíntia, Thiago, um beijo grande, obrigado pelos momentos de amizade. Ao pessoal do Movimento Galo 105 Minutos, o sentimento nunca vai parar! A alguns amigos que minha amizade está em hibernação, Miron, Alex, Macarrão, Claúdio, Pedro, Diana, Igor, Décio, Café, Honda, Paulo.

Aos professores Meira, Gordo, Fernando, Sérgio Campos, Márcio Bunte, Jussara, Virgílio, Dorgival, Mário, Jeroen, obrigado por todas as oportunidades, conselhos e avaliações. As vezes não entendemos a pressão que passamos, mas “a diferença entre o carvão e o diamante é a pressão". Ao pessoal do Speed e do DCC, Coutinho, Crocha, George, Túlio, Mribeiro, Lamarque, Charles, Carlos, Lídia, Kapão, FH, Hélio, Ceolin, Diego, Carol, Flip, Flop, Kraemer, Gimmes, Allan Jones, Don, Mister X, Jvictor, Andrec, Dineu, Itamar, Canguc/ssu, Rui, Silvio, Logan, PL, Heitor, Helen, Arbex, Bob, Cascardo, Arley, Elisa, Juliano, Euder, Fred, Tassni, Robert, Hugo, Fabrício, Fernando, Ismael, Zeniel, Luis, Delboni, Monique, Candian, Pedro, Rauber, Olga, Gusmão, Cazita, Emílio, Krusty, Edré e todos os que eu esqueci (sim, se você está lendo, você deveria estar aqui). Ao pessoal que cuida do departamento, Lazslo, Murilo, Renata, Túlia, Antônia, Alexandre, Geraldo.

Agradeço ao Chewbacca por cuidar da princesa Leia enquanto o Han Solo estava Congelado de posse do Boba Fett para ser entregue ao Jabba the Hutt. Que a força esteja com todos vocês!

"Existem três jeitos de fazer as coisas: o jeito certo, o jeito errado, e o meu jeito, que é igual ao jeito errado, só que mais rápido."
(Homer J. Simpson)

Resumo

As arquiteturas de computação são cada vez mais distribuídas e hierárquicas, ou seja, existem vários computadores num cluster, sendo que cada computador possui vários processadores e cada um deles possui vários núcleos. Diversos modelos foram propostos para explorar essas características das arquiteturas de computação, dentre esse modelos se destacam os modelos de computação em *Dataflow*, que acoplam-se bem à nova organização das arquiteturas distribuídas, pois permitem a descrição das aplicações como um conjunto de recursos que se conectam através de fluxos de dados.

Se as aplicações forem traduzidas automaticamente para esses modelos de computação em *Dataflow*, uma gama maior de aplicações poderá dispor de todos os benefícios de uma execução num ambiente distribuído e hierárquico.

O objetivo desse trabalho é avaliar estratégias de compilação para realizar a transformação de código para execução no ambiente de computação em *Dataflow* de forma automática, permitindo o uso desse modelo de programação por programadores que não dominem essa tecnologia.

Nessa dissertação é apresentado um protótipo de um tradutor *fonte para fonte* para realizar essa transformação. Esse tradutor recebe código em C que esteja na forma de um *loop canônico* e mapeia o código em filtros para execução no Anthill, uma implementação do modelo de computação em *Dataflow*. O foco do tradutor é gerar um código que seja capaz de executar em paralelo. Não há preocupação com a otimização do código gerado nem com possíveis gargalos gerados na leitura ou na escrita de dados na saída.

O tradutor separa cada bloco de execução e o insere num filtro Anthill, sendo que as dependências de dados existentes entre filtros são resolvidas através da insersão de uma comunicação entre eles. Se o filtro onde um dado é gerado iterar sobre o mesmo domínio de um outro filtro, a comunicação a ser realizada é o *labeled-stream* garantindo que o dado chegue a instância do filtro onde ele é necessário.

Foram avaliadas três aplicações de Mineração de Dados, o Frequent Itemsets, o K-means e o K-nearest neighbors. Utilizando as estratégias apresentadas nessa dissertação

foram gerados filtros e executados experimentos medindo o tempo de execução para cada um deles.

Os filtros gerados pelo tradutor mostraram resultados satisfatórios de *speed-up* e *scale-up* aproveitando os recursos disponíveis para a execução da aplicação. Além disso, foi comparada a execução dessa aplicações com versões implementadas por programadores experientes, e os resultados dos tempos de execução para os algoritmos Itemsets Frequentes e K-nearest neighbors gerados automaticamente foram bem próximos dos algoritmos gerados manualmente. O algoritmo K-means gerado automaticamente teve um desempenho (em termos de tempo de execução) bem inferior ao desenvolvido manualmente, mas ele suportou melhor o aumento do número de processadores disponíveis e teve até desempenho superior em alguns casos.

Em suma, o uso das estratégias de compilação apresentadas nesse trabalho pode permitir uma gama maior de aplicações aproveitando melhor os recursos de arquiteturas de computação que sejam distribuídas e hierárquicas e pode permitir o uso maior de ambientes de programação em *Dataflow* para a solução de problemas de computação.

Abstract

The computers architectures are becoming more distributed and hierarchical,that is, there exist lots of computers in a cluster and there are many processors in these computers and these processors have multiple cores.

Many models have been proposed to explore the features of the computers architectures. The Data-flow computing models stand out among the proposed models because they can be better mapped into these architecture allowing applications to be described as a set of resources connected with data-flow streams.

If the applications are automatically translated to these data-flow computing models, an wider range of applications can benefit from the execution in an distributed and hierarchical environment.

The goal of the present work is to evaluate compilation strategies to do the automatic transformation of code for execution in a data-flow computing environment, allowing the usage of this model for programmers that don't need to understand this technology.

In this work a source-to-source translator prototype is presented to do the automatic transformation. This translator receives C code in a canonical loop shape and map the code into filters for executing in Anthill, an implementation of the data-flow computing model. The translator focuses in the parallelization of the application. There is no concern about code optimization and with the quality of reading and writing data speed.

The translator splits each execution block of the code and inserts into a new Anthill filter, all the data dependency that exists between two filters are solved by inserting a stream for communication of this data. If the generated filter iterates over the same domain of another filter, the communication inserted is a *labeled-stream* that will guarantee the data available where it is needed.

Three Data Mining applications have been analyzed in the work, Frequent itemsets, K-means and K-nearest neighbors. Filters have been automatically generated and experiments have been conducted to measure the execution time of these applications.

The translator generated filters achieve good results in terms of speed-up and scale-up, using the available resources for their execution. These applications have been compared with manual implementations made by experienced programmers. The results of both versions (automatically and manually transformed) from Frequent Itemsets and K-nearest neighbors had similar performance. The automatic generated version of K-means have worse performance (it terms of execution time) compared with the manual version, but with a larger number of processors, the automatic version had better performance in some cases.

The usage of the compilation strategies presented in this work allow an wider range of applications have better usage of the computer architecture resources and allow the usage of Data-Flow computing models to solve more problems in computer science.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Computação com múltiplos núcleos tem se tornado o padrão da indústria em busca de alto desempenho, exemplos de computadores com essas características são o Sun Ultra Sparc T1 [Kongetira et al., 2005], Sony/IBM/Toshiba Cell [Hofstee, 2005] e a NVIDIA GeForce 8800 GTX [Ryoo et al., 2008]. Além disso, arquiteturas de múltiplos núcleos estão sendo utilizadas em computadores pessoais, com a Intel e AMD fabricando processadores com 2, 4 ou mais núcleos. Para atingir alto desempenho, são utilizadas arquiteturas com vários computadores de múltiplos núcleos interligados via rede formando um *cluster* e num nível superior, vários *clusters* conectados formando um *grid* [Foster et al., 2002].

A proliferação de arquiteturas que permitem o processamento paralelo trouxe uma revolução para a área de linguagens de programação, conforme descrito em [Hall et al., 2009], "...Os próximos 50 anos da pesquisa em compiladores serão devotados para a geração e verificação de programas paralelos...". Como ilustração, pode-se citar o fato de que 17 dentre 34 artigos publicados nos anais da conferência *ACM's Conference on Programming Languages, Design and Implementation* [Gupta & Amarasinghe, 2008] de 2008 lidavam com problemas relacionados à computação paralela.

Modelos de computação em Data-flow [Johnston et al., 2004, Whiting & Pascoe, 1994, Morrison, 1994] são organizados de forma hierárquica e se adaptam bem a essas arquiteturas. Nesses modelos, a computação é dividida em pequenos blocos de processamento que são conectados por fluxos de dados. Os dados atravessam o *pipeline* desses blocos de processamento sendo modificados por cada um dos estágios. A ativação de um bloco de processamento se dá no momento em que um dado de entrada chega. Se determinado dado for necessário em um estágio, há um caminho para o dado transitar do estágio onde ele é gerado, até o estágio onde ele

é utilizado. Um conjunto de blocos de processamento também pode ser conectado a outro conjunto de blocos de processamento, formando a hierarquia de processamento. Sendo assim, os modelos de computação em Data-flow se acoplam facilmente nas arquiteturas de computação pois há um mapeamento natural entre ambos.

Esse mapeamento natural só é possível se o programador utilizar um modelo de computação em Data-flow para descrever a aplicação desejada, mas isso demanda o conhecimento do modelo e das suas construções. Em alguns casos é necessário a compreensão de problemas de sincronização, bem como chamadas para realizar a comunicação de dados.

Como nem todos os programadores são capazes de descrever as aplicações em modelos de computação em Data-flow, as aplicações são escritas sem aproveitar todas as características desses ambientes. Se as aplicações forem traduzidas automaticamente para esses modelos de computação em Data-flow, uma gama maior de aplicações poderá dispor de todos os benefícios de uma execução nesses ambientes.

O modelo implementado pelo Anthill [Ferreira et al., 2005] é um modelo de computação em Data-flow no qual a computação é divida em estágios, chamados filtros, que se comunicam através de fluxos (um fluxo liga a saída de um filtro na entrada de outro filtro). Cada um dos estágios pode ser replicado, criando um grafo parecido com um DAG onde uma comunicação pode acontecer entre qualquer um dos estágios sem restrições, exceto pelo fato de permitir a criação de ciclos. O dado comunicado pode possuir qualquer tipo. Os métodos de comunicação disponíveis para o Anthill são o *broadcast*, no qual todas as instâncias de um filtro recebem os dados, *round-robin*, no qual há um revezamento entre as instâncias e *label-stream* onde um rótulo da mensagem indica para qual instância de filtro ela deve ser enviada.

O desafio do uso do Anthill é implementar os algoritmos utilizando o modelo filtro-fluxo, que assim como em outros modelos de computação em Data-flow, há uma série de características que precisam ser consideradas para inserir uma comunicação e para dividir um espaço de iteração de um *loop* entre dois filtros.

1.1 Objetivo

O objetivo desse trabalho é avaliar estratégias de compilação para realizar a transformação de código para execução no ambiente de computação em Data-flow, de forma automática, permitindo o uso desse modelo de programação por programadores que não dominem essa tecnologia. Nessa dissertação é apresentado um protótipo de um tradutor *fonte para fonte* para realizar essa transformação, que será chamado apenas

de tradutor no texto. Para efeito de demonstração será utilizada uma representação em ML dos algoritmos e das transformações realizadas pelo tradutor.

1.2 Contribuições

O tradutor assume que o código escrito pelo usuário recebe a entrada de dados através de uma fluxo do Anthill e deve escrever os dados de saída num fluxo do Anthill. Se desejar, o usuário pode apresentar o código de leitura e escrita de dados assinalado no código permitindo que o tradutor gere uma versão desses 2 filtros.

Foram avaliadas nesse trabalho três aplicações de mineração de dados bastante conhecidas e exploradas, o K-nearest neighbors [Witten & Frank, 2005], Frequent Itemsets [Agrawal et al., 1993] e K-means [Macqueen, 1967]. Conforme será visto no capítulo 4 as aplicações que foram avaliadas nesse trabalho podem ser mapeadas num *loop canônico*.

O *loop canônico* é um *loop* de execução que contém uma série operações de *Map-reduce* que realizam transformações nos dados de entrada e enviam o dado modificado para a próxima operação de map-reduce. Cada uma dessas operações é mapeada em um filtro do Anthill que são conectados através de fluxos criados para suprir as dependências de dados. Ele pode ser visto no algoritmo 1.

Algoritmo 1: – **Loop Canônico**:

1: $L_1 \leftarrow \ldots$
2: **while** ... **do**
3: $L_2 \leftarrow MR_2(L_1)$
4: $L_3 \leftarrow MR_3(L_2)$
5: ...
6: $L_n \leftarrow MR_n(L_{n-1})$
7: $L_1 \leftarrow L_n$
8: **end while**

Externamente o algoritmo possui a construção *while* (enquanto) que itera até que o resultado a ser calculado fique estável, o número de iterações alcance uma quantidade máxima ou a tolerância mínima de erro no resultado seja alcançada. Em cada uma de suas iterações são verificados todos os elementos do conjunto de entrada e alguma transformação é realizada em torno desses elementos. As três aplicações citadas anteriormente, assim como outras aplicações de mineração de dados, carregam em comum as características de *loop canônico*.

Inicialmente o algoritmo recebe um trabalho ($Work$) para executar. Esse trabalho contem as informações necessárias para leitura dos arquivos de entrada e os parâmetros

para execução do algoritmo. A parte de leitura de dados e preparação dos parâmetros foi omitida para melhorar a compreensão do *loop* canônico.

Cada operação de Map-Reduce é identificada e inserida em um filtro. A comunicação entre os filtros é definida a partir da cadeia de uso-definição de variáveis. Para cada uso da variável no filtro, a última definição é encontrada e se essa estiver em outro filtro, deve-se realizar a comunicação. Para cada tipo de dado a ser comunicado é criado um fluxo do Anthill. Dependendo de como a variável é utilizada, opta-se por utilizar *broadcast + round-robin* ou *labeled-stream*. O *labeled-stream* é utilizado quando os filtros que iteram sobre um vetor estão no mesmo domínio e não há dependência. O domínio de iteração do *loop* é dividido pelo número de instâncias dos filtros e utiliza o índice como rótulo. Em alguns casos o filtro deve receber mensagens de todas as instâncias do filtro que comunica e agregar essas mensagens, dependendo do caso, fazendo a *redução*.

O código de entrada *pode* possuir diretivas que auxiliem o tradutor a realizar seu trabalho, sendo tais diretivas responsáveis por apresentar as características do programa ao tradutor.

Parâmetros e variáveis que são definidas antes do *loop* principal são utilizadas para preencher uma mensagem que é enviada a todas as instâncias de todos os filtros no início do processamento, o chamado *Work*. Para implementar o protótipo do tradutor apresentado nessa dissertação optou-se pelo uso do *Cetus* [Lee et al., 2003, Johnson et al., 2004], desenvolvido em *Java*. Ele foi modificado de acordo com o que foi apresentado nessa dissertação para transformar código para o Anthill.

Parte dos resultados apresentados nessa dissertação foram publicados no XIII Simpósio Brasileiro de Linguagens de Programação [da Mata et al., 2009].

1.3 Organização

No capítulo 2 são apresentados os trabalhos relacionados, incluindo uma descrição mais detalhada do modelo implementado pelo Anthill. O processo de compilação desenvolvido é apresentado no capítulo 3. As aplicações usadas para avaliar o processo de compilação são apresentadas no capítulo 4, e os resultados da avaliação estão no capítulo 5. A conclusão é mostrada no capítulo 6. No apêndice A são apresentados em mais detalhes os algoritmos de compilação e no apêndice B são apresentados os algoritmos utilizados na entrada e gerados na saída do tradutor.

Capítulo 2

Trabalhos Relacionados

Nesse capítulo são apresentados os trabalhos relacionados a esse trabalho e a contribuição dessa dissertação para cada um deles. A seção 2.3 é dedicada exclusivamente a apresentar o *Anthill*, o ambiente para execução de algoritmos de computação em *Data-flow* que foi utilizado nessa dissertação.

2.1 Modelo filtro-fluxo

No Modelo de computação em *Data-Flow* [Johnston et al., 2004] o algoritmo é dividido em estágios de computação que são ligados através de caminhos por onde o dado transita. A computação em Data-flow é definida com um conjunto de primitivas e um modelo de programação para desenvolvimento de aplicações como um grafo direcionado, onde as arestas representam os fluxos de dados, e os nós as operações aplicadas aos mesmos. O dado caminha através das arestas sendo modificado nos nós de computação. A idéia de se desenhar o algoritmo ao invés de programá-lo foi apresentada em [Sutherland, 1966], sem utilizar o nome de modelo de computação em *Data-Flow*. Em outros trabalhos [Johnston et al., 2004, Whiting & Pascoe, 1994, Morrison, 1994, Gurd et al., 1985], o modelo foi detalhado e as idéias foram consolidadas, mas existem alguns modelos de programação que não estavam incluídos por seus autores nessa categoria, como é o caso do Anthill.

Outra nome da computação em Data-flow é a programação em fluxos (*stream programming*) [Spring et al., 2007], nesse modelo os programas são uma composição de filtros que se comunicam via canais uni-direcionais. Cada filtro é uma unidade de

processamento que pode ou não, manter um estado interno. Os dados são lidos de um canal de entrada e escritos num canal de saída.

O modelo de programação utilizado pelo *Anthill* é uma extensão do modelo *filter-stream* ou *filtro-fluxo* [Beynon et al., 2000, Beynon et al., 2001] que pode ser visto dentro do modelo de computação em *Data-Flow*. Nesse modelo as unidades de computação são chamadas de *filtros* e a comunicação é feita através de fluxos (*streams*), que definem a conexão lógica entre 2 filtros (A e B por exemplo). Na seção 2.3 esse modelo será descrito.

2.2 Map-Reduce

Nessa dissertação as operações de Map-Reduce serão utilizadas para representar os algoritmos e apresentar de uma forma mais didática as transformações realizadas.

Uma operação de Map-Reduce é a combinação de uma operação que mapeia os elementos de um conjunto em outro conjunto (map) e uma operação que realiza a redução dos dados (reduce) [Morita et al., 2007, Akimasa et al., 2009], pode-se citar o framework do Google [Dean & Ghemawat, 2004] para computação distribuída como exemplo de ambiente de programação de Map-Reduce.

A função *map* aplica a função f em cada um dos elementos do conjunto de entrada:

$$map\ f\ [x_1, x_2, \ldots, x_n] = [f\ x_1, f\ x_2, \ldots, f\ x_n]$$

A função *reduce* realiza a redução dos dados aplicando seguidamente o operador binário associativo $\odot$ nos elementos do conjunto de entrada:

$$reduce\ \odot\ [x_1, x_2, \ldots, x_n] = x_1 \odot x_2 \odot \ldots \odot x_n$$

As operações de Map e Reduce definidas anteriormente são esqueletos paralelos, ou *parallel skeletons* [Cole, 1988], como a operação $\odot$ é associativa, combinações dessas operações podem ser implementadas em paralelo.

Os artigos [Gibbons, 1996] e [Morita et al., 2007] mostram como derivar automaticamente a versão paralelizada de um algoritmo que esteja escrito na forma de Map-Reduce, derivando automaticamente o operador de redução. Apesar de ser um resultado interessante, e que facilita a paralelização automática de aplicações, essa dissertação assume que o operador de redução é associativo, sendo que a grande contri-

buição do trabalho é como realizar a distribuição das operações entre os processadores e como realizar a comunicação de forma eficiente.

O artigo [Ponnusamy et al., 1993] mostra como técnicas de compilação podem ser mescladas com técnicas de análise em tempo de execução para solucionar, com bom desempenho, problemas de comunicação que não seguem padrões bem definidos, também chamadas de comunicações irregulares, ou que dependem dos dados de entrada. Os autores apresentam o método de derivação de um grafo que contem informações necessárias para descobrir a melhor distribuição de dados num programa. Essa abordagem será avaliada em trabalhos futuros.

A paralelização de operações de redução foi abordada em [Hall et al., 1996]. Esse artigo mostra que uma análise inter-procedural pode ser realizada de forma eficiente e apresentar resultados satisfatórios para paralelização automática em máquinas multiprocessadas, apesar de não ser capaz de resolver todos os problemas (alguns requerem uma análise em tempo de execução). A diferença principal desse trabalho com o apresentado nessa dissertação está no fato de realizar paralelização em um ambiente multi-processado com memória compartilhada, ao contrário do tradutor apresentado que foca na paralelização de operações de map-reduce em um ambiente sem memória compartilhada.

A detecção de variáveis de redução é abordada em [Han & Tseng, 1999, Rauchwerger & Padua, 1999, Hall et al., 1995], porém essa dissertação foca na realização de otimização em operações que já foram identificadas como reduções, deixando esse tipo de detecção para ser implementada em trabalhos futuros.

2.3 Descrição do *Anthill*

O *Anthill* implementa o paradigma de processamento *filter-stream* onde as unidades de programação são chamadas de *filtros* e os dados são trocados entre eles através dos *fluxos* (*streams*). Essa conexão lógica entre os filtros é uni-direcional e os dados são transferidos do *filtro produtor* para o *filtro consumidor*. Os dados não possuem tipo pré-definido. Cada *fluxo* possui três tipos de comunicação possível:

- *broadcast*: Nesse tipo de comunicação uma instância de um filtro envia mensagem a todas as instâncias do outro filtro.

- *round-robin*: Nesse tipo de comunicação uma instância de um filtro envia cada mensagem para uma instância diferente do outro filtro, realizando um revezamento. esse padrão é repetido.

- *label-stream*: Nesse tipo de comunicação a mensagem a ser enviada recebe um rótulo e ela é roteada para a instância o outro filtro através de uma função hash que mapeia rótulos em instância de filtros.

Diferentemente do Datacutter [Beynon et al., 2000, Beynon et al., 2001], que é voltado para paralelismo de grão grosso com troca de mensagens com maior quantidade de dados, o Anthill é voltado para o paralelismo de grão fino, permitindo explorar mais níveis de paralelismo.

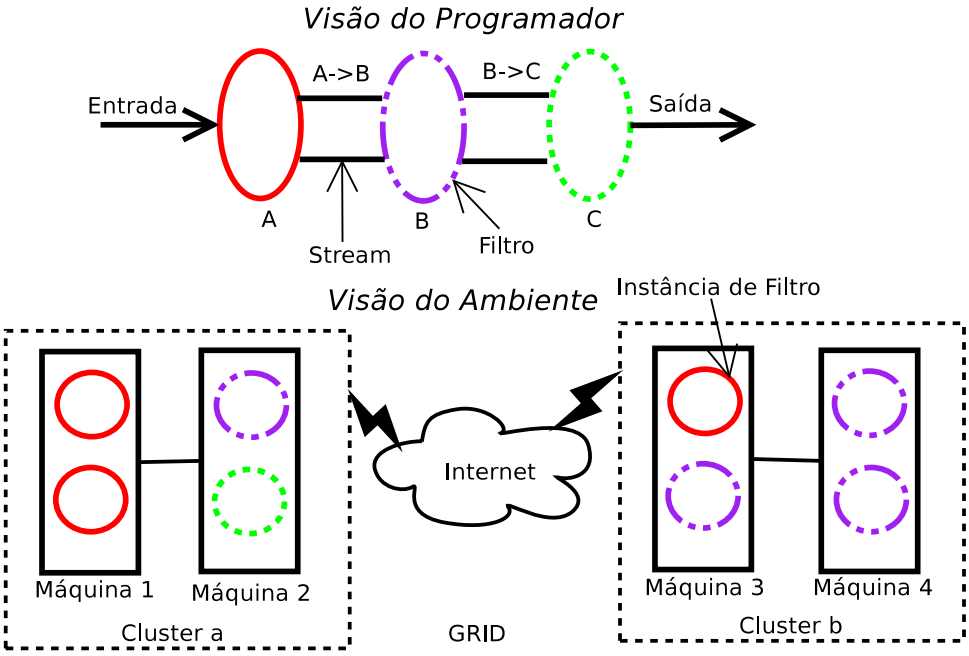


Figura 2.1: Paradigma *filter-stream* - Visões do programador e do ambiente

Nesse modelo, a entrada dos dados pode ser dividida em pequenas partes, chamadas de *units of work* (*UOWs*) que podem ser trabalhadas de forma simultânea pelo grupo de filtros como num *pipeline* (veja a visão do programador na figura 2.1).

Visando melhorar o paralelismo de dados, o ambiente de execução Anthill provê um mecanismo chamado *cópia transparente de filtros*. Nele um filtro pode possuir várias instâncias em qualquer outra máquina sem que o resultado final da computação seja alterado. A cópia é chamada de transparente porque as instâncias de um mesmo filtro compartilham as mesmas conexões lógicas de entrada e saída com os outros filtros, ou seja, compartilham os mesmos *streams*. Uma cópia transparente de um filtro pode ser feita se a computação total gerada não for alterada, ou seja, se a entrada de *UOWs* em instâncias diferentes de um mesmo filtro não alterarem as saídas desses filtros.

O ambiente mantém a ilusão do uso de uma única conexão lógica entre dois filtros, o produtor e o consumidor. Essa ilusão só é possível porque o ambiente faz o escalonamento dos elementos de saída de filtros utilizando políticas como *round-robin* (*RR*), *broadcast* (*BC*) e *labeled-stream* (*LS*) que vão selecionando instâncias de filtros para o envio desses elementos.

O *labeled-stream* é uma política que escolhe a instância do filtro que irá receber a mensagem através de um rótulo (*label*) da mensagem. Essa escolha é feita com base em uma função *hash* que o programador pode criar, ou utilizar a existente no ambiente. Essa política permite que, caso haja uma relação entre dois dados a serem enviados, essa relação pode ser utilizada para atribuir o rótulo e assim garantindo que eles cheguem à mesma instância de filtro para o qual a comunicação é realizada, permitindo um particionamento dos dados de forma controlada.

A figura 2.1 mostra como é a visão do programador que utiliza esse paradigma para programar e como o ambiente mantém o processamento espalhado por todo o *grid*. Nesse exemplo, o programador cria os filtros A, B e C; e cria *streams* entre os filtros A e B e entre B e C. O programador não precisa se preocupar como o número de instâncias dos filtros ou se eles serão instanciados em máquinas diferentes. O ambiente toma a decisão, nesse caso, de criar 3 instâncias do filtro *A* (sendo duas na máquina 1 e uma na máquina 3), 4 instâncias do filtro B (sendo uma na máquina 2, uma na máquina 3 e duas na máquina 4) e uma instância do filtro C (na máquinas 2), sendo essas decisões tomadas na tentativa de promover o balanceamento carga e uma melhor utilização dos recursos de rede.

Um ambiente como esse provê, portanto, paralelismo de tarefas, pois se pode dividir a computação total em filtros e também provê paralelismo de dados, pois se pode criar cópias transparentes de filtros que irão trabalhar de forma independente os dados que receberem, além é claro de métodos de comunicação poderosos como o *labeled-stream*.

Uma implementação orientada a eventos desse ambiente foi apresentada em [Teodoro et al., 2008]. Nessa implementação os filtros são ativados sob-demanda quando os dados ficam disponíveis para processamento, mas essa dissertação não foca na geração de código para o modelo orientado a eventos desse ambiente, ficando essa idéia para um trabalho futuro.

2.4 Paralelização Automática

Os artigos [Hall et al., 1995, So et al., 1998] mostram as estratégias de paralelização automática de *loops* utilizadas pelo SUIF. Os artigos mostram as estratégias necessárias para descobrir as formas de se executar iterações de *loops* de forma paralela. Realiza-se a privatização[1] de vetores para a execução em multi-processadores e identifica quais *loops* podem ser executados de forma paralela. Essas propostas não consideram paralelismo que envolve processos comunicantes, diferenciando, portanto, da proposta dessa dissertação de mestrado.

O UPC ou **U**nified **P**arallel **C** [Consortium, 2005] propôs uma extensão de *C* que se baseia no paradigma *SPMD* (*Single Program Multiple Data*), onde um único código é executado em todos os componentes da plataforma de execução (conhecidos a priori), sendo o dado dividido entre estes. Ela implementa um modelo de programação uniforme tanto para memória distribuída quanto compartilhada. Nela o programador deve apresentar no código as variáveis que podem ser lidas ou escritas por um determinado processador. O modelo utilizado nessa dissertação divide a computação em vários estágios. Apesar do UPC ser uma linguagem bem definida para descrever código paralelo, optou-se por utilizar na entrada do tradutor código mais simples possível, sem construções como barreiras, diretivas de comunicação de dados, entre outros, por isso não foram utilizadas linguagens como essas. Essa decisão foi tomada para permitir o uso de código desenvolvido por programadores inexperientes, sem a necessidade do uso de uma linguagem externa.

Outra linguagem para definição de programas paralelos é o Titanium [Yelick et al., 1998], um dialeto de *Java* desenvolvido para aplicações de computação paralela científica que possui definições para barreiras e sincronização, mensagens ponto a ponto e *broadcast*; variáveis privatizadas; *loops* cuja ordem das iterações não é pré-definida (*foreach*); e tipos para definição de matrizes . O compilador dessa linguagem (que foi desenvolvido em *C++*) é capaz, através da descrição desses elementos, de gerar código *C* para a aplicação analisada. Como o objetivo dessa dissertação é realizar a paralelização automática de aplicações sem o uso de diretivas especiais, optou-se por não utilizar essa linguagem.

O Impact [Chang et al., 1991] e o Open Impact [mei Hwu, 2006] são projetos criados para desenvolver código de qualidade para arquiteturas que permitem paralelismo em nível de instruções. Essa dissertação difere-se desses trabalhos pois está focada no paralelismo em alto-nível, e não em nível de instruções.

[1] Privatização de variáveis é o processo de realizar cópias locais de variáveis nos vários processos participantes de uma computação, de forma a isolar o valor local das variáveis.

Um modelo matemático que combina várias técnicas de descoberta de possibilidades de paralelismo é apresentado em [Lim et al., 1999]. Esse modelo é capaz de descobrir a distribuição de código entre processadores de um sistema multiprocessado de forma a maximizar o grau de paralelismo e minimizar o número de mensagens comunicadas. Ele se baseia na execução do mesmo código em vários processadores e na diminuição da sincronização entre cada um dos processadores. As idéias apresentadas nesse artigo não contemplam fatores como o uso de comunicação baseada nos domínios de iteração dos *loops*. Em trabalhos futuros pretende-se utilizar as estratégias apresentadas nesse artigo para detectar a possibilidade de unificar filtros reduzindo a quantidade de comunicação.

O artigo [Ferreira et al., 2000] apresenta técnicas para geração de código automática para execução de aplicações que processam grandes volumes de dados. Esse artigo é o início dos trabalhos [Du & Agrawal, 2004, Du et al., 2003] que fazem o mapeamento dessas aplicações intensivas em dados nos filtros do modelo *filter-stream* [Beynon et al., 2000, Beynon et al., 2001]. Nesses trabalhos são feitas diversas análises do código em tempo de compilação e execução de forma a sinalizar o mapeamento em filtros.

Os trabalhos [Du et al., 2003, Du & Agrawal, 2005], que descrevem o processo de geração de filtros para o modelo *filter-stream* realizando a divisão de algoritmos descritos num dialeto baseado em *Java*, utilizam heurísticas para avaliar a quantidade de comunicação realizada e estimar qual o melhor particionamento a ser realizado. Os algoritmos propostos por esses trabalhos obtiveram bons resultados na geração automática dos algoritmos, mas os autores focam na divisão do algoritmo e não nas diferentes possibilidades de padrões de comunicação a serem utilizadas. Os autores desses trabalhos dividem o algoritmo a ser particionado nas menores unidades possíveis, espalhando-as em filtros de forma a minimizar fatores como tempo de execução, quantidade de dados comunicados, entre outros. Esses artigos apresentam abordagens que podem ser utilizadas nos trabalhos futuros dessa dissertação.

Várias aplicações de mineração de dados podem utilizar o mapeamento proposto nessa dissertação. O artigo [Li et al., 2003] mostra o uso de técnicas de compilação para geração automática código paralelizado, apresentando os resultados para aplicações de mineração de dados. Os resultados dos experimentos mostraram a viabilidade da paralelização desse tipo de aplicação, além de mostrar que técnicas conhecidas de otimização de código podem ser utilizadas para melhorar os códigos em relação aos paralelizados manualmente. O código a ser paralelizado deve conter assinalamentos marcando as posições onde cada filtro começa, quais são as dependências de dados e como esses dados devem ser comunicados, conduzindo o compilador para realizar a

paralelização da melhor forma possível. Esse tipo de paralelização é chamado de semi-automático, pois a tradução em outra linguagem é automática, mas a análise é feita manualmente.

Segundo o artigo [Góes et al., 2005] o modelo *filter-labeled-stream* [Ferreira et al., 2005] pode ser utilizado como modelo de programação dos códigos automaticamente gerados. Segundo os autores desse trabalho, se pode utilizar o grafo de tarefas (também chamado de grafo de dependência de dados) que representa a aplicação para descobrir quais são os filtros, qual o conteúdo de cada um deles e qual a comunicação a ser utilizada entre eles. O trabalho apresentado nessa dissertação pretende avaliar outras formas de se realizar a geração de código automaticamente encontrando padrões Map-Reduce na aplicação a ser paralelizada.

Na dissertação de mestrado [de Castro Santos, 2006] o autor apresenta uma estratégia para geração automática de código para o *Anthill*, mas para realizar essa geração o código precisa ter diretivas para informar como os particionamentos e comunicações devem ser realizadas. A abordagem apresentada aqui é diferente pois, apesar de também levar em consideração o grafo de dependências de dados, o mapeamento em filtros depende da identificação de padrões no formato Map-Reduce e é feito de forma automática.

Para realizar o desenvolvimento das transformações de código propostas por esse trabalho, optou-se por utilizar um compilador já existente e assim aproveitar a sua estabilidade e maturidade. A escolha desse compilador foi realizada através da avaliação de ambientes de desenvolvimento de compiladores. Buscou-se encontrar um ambiente que possuísse características que permitissem a rápida prototipagem das propostas dessa dissertação, o uso de código já existente na entrada do compilador e a possibilidade de utilização futura desse protótipo. As seguintes características foram consideradas: código fonte aberto, representação interna do código analisado em linguagem de alto-nível, transformação de seu código interno em *C* legível (mantendo *loops for, while e do*), rápida implementação de modificações no código, *curva de aprendizagem* rápida da estrutura do compilador ,além de capacidade de reutilização futura dos pacotes desenvolvidos.

Para implementar o protótipo do tradutor apresentado nessa dissertação optou-se pelo uso do Cetus [Lee et al., 2003, Johnson et al., 2004], que é um compilador para *C, C++ e Java* criado para gerar código *C* em sua saída. Ele possui uma representação intermediária que é capaz de gerar código fiel àquele apresentado na entrada (para programas em *C*). Modificações simples podem ser implementadas facilmente, desde que o programador que as fará esteja ciente de estar lidando com uma linguagem intermediária tão complexa quanto o *C*. O Antlr [Parr & Quong, 1995], que é um parser para

linguagens de programação que aceita gramáticas *LL(k)* e é capaz de reconhecer várias linguagens (*Java, C#, C++, C, Python*) foi escolhido como *frontend* de compilação. No apêndice A é apresentado um apanhado dos compiladores avaliados para realizar essa escolha.

2.4.1 Sumário

Nesse capítulo foram apresentados os trabalhos relacionados a essa dissertação. Foram apresentados compiladores que realizam paralelização automática de código, o padrão Map-Reduce que é identificado no código a ser paralelizado e o paradigma *filtro-fluxo* que é implementado pelo Anthill, o ambiente de computação paralela para o qual o tradutor gera o código paralelizado.

Capítulo 3

Metodologia

Esse capítulo é dividido em 2 partes, na primeira será apresentada duas paralelizações diferentes para o problema de divisores frequentes para demonstrar a importância da escolha correta da estratégia de comunicação. Na segunda parte as estratégias de compilação utilizadas são abordadas.

3.1 Como paralelizar?

Existem várias formas de se paralelizar uma aplicação, sendo que cada estratégia diferencia-se da outra na forma como os dados e as tarefas são particionados entre os processadores. Geralmente o tamanho da entrada deve ser levado em consideração para determinar a melhor forma de se paralelizar uma aplicação. Esse fato é ilustrado através da comparação de duas estratégias diferentes para o problema de *divisores frequentes*, que é definido da seguinte forma:

- Problema de Divisores Frequentes
  Instância: Um conjunto a de *atributos inteiros*, os divisores; um conjunto t de *transações inteiras*, os dividendos, e um inteiro σ, chamado de *suporte*.
  Problema: encontrar o conjunto $f \subseteq a$, tal que $n \in f$ se, e somente se, n divide (exatamente)σ elementos ou mais de t.

A figura 3.1 mostra como são as duas estratégias, na letra (a) os atributos são divididos entre processos e na letra (b) as transações são divididas entre os processos. Nas seções a seguir serão descritas essas duas estratégias.

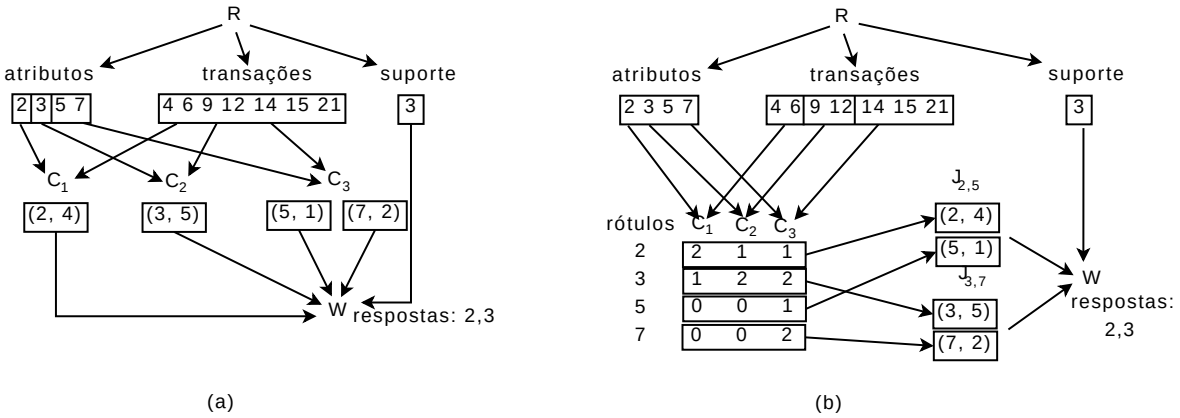


Figura 3.1: Paralelizando o problema de Divisores Frequentes. (a) O **Leitor** (R) divide os atributos entre os **Contadores** (C), que envia o resultado final ao **escritor** (W). (b) O **Leitor** particiona as transações entre os **Contadores**, que enviam os resultados parciais aos **mescladores** (J) através de *label-stream*

3.1.1 Particionando atributos entre processos

Uma forma natural de paralelizar o problema é dividir os atributos entre os processos, nesse caso, cada processo recebe todas as transações e apenas uma parte dos atributos. Essa idéia está ilustrada na figura 3.1 (a). São considerados 3 tipos de filtros:

Leitor, para distribuir dados entre os processos;

Contador, para somar o número de vezes que cada atributo divide exatamente as transações;

Escritor, para imprimir o resultado final.

O *Leitor* envia transações utilizando *broadcast* e envia os atributos via *round-robin*

Se c é o conjunto de **Contadores** disponíveis, então cada **contador** realizar $|a|/|c| \times |t|$ operações de divisão. O **Leitor** envia para cada **contador** $|a|/|c| + |t|$ mensagens. O total de mensagens enviados do **Leitor** para todos os **Contadores** é, portanto, $|a| + |c| \times |t|$.

3.1.2 Particionando transações entre processos

Se o número de transações é muito maior que o número de atributos, então a paralelização adotada na seção 3.1.1 poderá gerar um número grande de mensagens entre o **Leitor** e os **Contadores**. Num ambiente onde comunicação é um recurso caro, a aplicação pode acabar gastando mais tempo na troca de mensagens do que na computação do resultado efetivamente. Nesse caso é melhor dividir as transações entre os **Conta-**

dores. Os fluxos do tipo *label-stream* providos pelo *Anthill* facilitam na implementação desse tipo de particionamento.

Nessa nova abordagem, mostrada na figura 3.1(b), um quarto tipo de filtro foi criado, chamado de **Mesclador**. **Contadores** recebem todos os atributos através de *broadcast* do **Leitor**. O **Leitor** também envia, através de um fluxo *round-robin*, parte das transações para cada um dos **Contadores**.

O **Contador** computa tuplas $(a_i, n), a_i \in a$, onde n é o número de transações que a_i espera dividir. Ao finalizar todas as computações, o **Contador** envia as tuplas para os **Mescladores**. Essas tuplas são enviadas através de um canal *label-stream*, onde o rótulo é o próprio atributo a_i. Dessa forma cada mensagem (a_i, n) correspondente ao atributo a_i será enviada para o mesmo **Mesclador**, que será responsável por somar todos os n's e entregar o resultado final ao **Escritor**.

Como no caso anterior, o número total de operações realizadas por todos os **Contadores** é $|t| \times |a|$, sendo que cada um deles executa $|t|/|c| \times |a|$ divisões. Além disso, cada **Mesclador** realiza $|c| \times |a|/|j|$ adições na média, onde j é o conjunto de **Mescladores**. O número total de mensagens enviadas do **Leitor** para os **Contadores** é $|a| \times |c| + |t|$.

3.1.3 Análise quantitativa das duas estratégias

Cada uma das estratégias pode ser utilizada na geração de filtros, mas não sabe-se a priori qual delas deve ser escolhida. A forma de se escolher uma delas depende do tamanho da entrada.

A figura 3.2 mostra seis experimentos comparando as duas abordagens. Em cada experimento o número de **Contadores** usado é diferente. A abordagem que utiliza *label-stream* sempre utiliza 2 Mescladores. O número de transações é fixo, variando o número de atributos. Cada gráfico mostra o tempo de execução da aplicação no eixo y e o tamanho do conjunto de atributos no eixo x.

Se o universo de atributos possíveis é muito menor que o número de transações, a segunda abordagem, baseada no *label-stream* é a mais indicada, pois o número de mensagens trocadas é muito menor, ou seja, o *overhead* de fazer o *broadcast* de todo o conjunto de transações para todos os **Contadores** na primeira abordagem causa um maior impacto negativo no desempenho das aplicações do que se comparamos com a inserção de filtros **Mescladores** extra da segunda abordagem. À medida que o número de atributos cresce, e se aproxima do número de transações, a primeira abordagem começa a se tornar mais atrativa. Para conjuntos de 10^5 transações e mais de 8×10^4

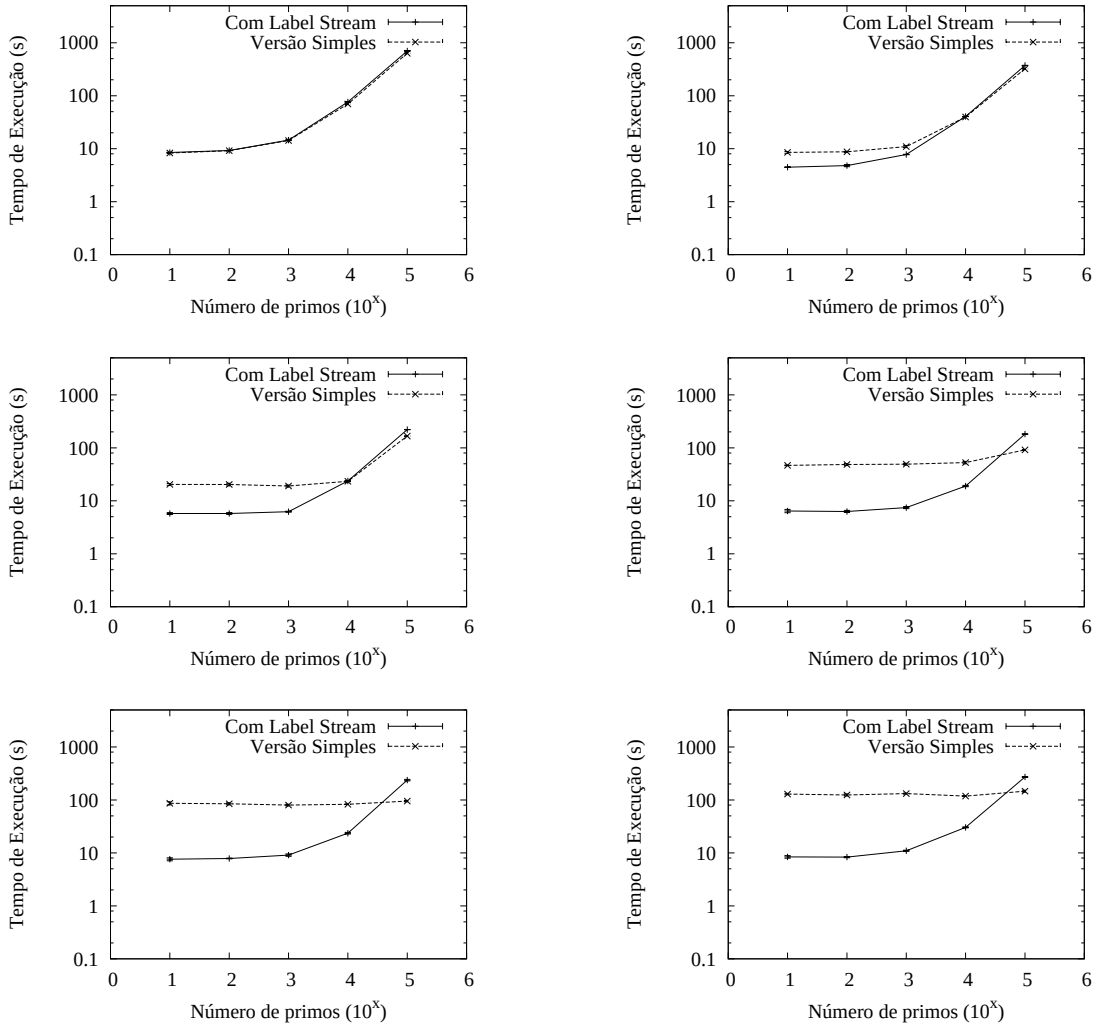


Figura 3.2: Comparando as duas versões paralelas do problema de divisores frequentes. O número de **Contadores** é variado nos experimentos. De cima para baixo, da esquerda para direita, são usados 1, 2, 4, 8, 12 e 16 Contadores, respectivamente. A versão com *label-stream* utiliza 2 filtros Mescladores. Foram utilizados 10^5 transações variando-se o número de atributos. Cada filtro executa em uma máquina separada.

atributos, a primeira abordagem supera a segunda. O objetivo dessa dissertação é tentar inferir a melhor divisão em filtros de forma automática.

3.2 Tradutor apresentado

Conforme apresentado em [Wolfe, 1996], um compilador é dividido em 4 partes, o *Front End*, as *Otimizações de Alto Nível*, as *Otimizações de Baixo Nível* e a *Geração de Código*, conforme pode ser visto na figura 3.3 retirada desse livro.

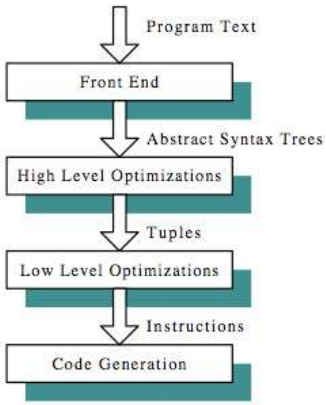


Figura 3.3: Estágios de um Compilador

O foco dessa dissertação é realizar as *Otimizações de Alto Nível* relacionadas com a transformação automática para execução no Anthill. Ao invés de realizar outras otimizações e a geração do código binário, o tradutor imprime o código em C novamente já com as transformações em filtros realizadas. Dessa forma é possível utilizar um outro compilador para realizar as outras otimizações desejadas e a geração dos binários para a execução na arquitetura desejada.

A figura 3.4 mostra os passos que são realizados pelo tradutor. A partir de um código sequencial o tradutor extrai as informações e as combina para gerar os filtros do *Anthill*.

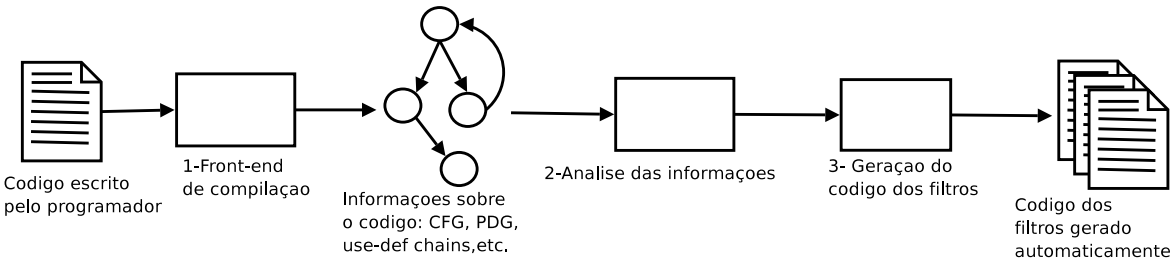


Figura 3.4: Tarefas do Tradutor

Além de estar no formato de *loop* canônico apresentado, o código a ser paralelizado deve conter algumas características importantes. O tradutor possui uma postura conservadora, ou seja, assume-se que o código de entrada está no formato correto e que o programador está ciente das transformações que são realizadas pelo tradutor. Dessa forma o protótipo pode apenas mostrar uma mensagem avisando que não há

uma paralelização possível quando casos de contorno acontecerem.

A condição de parada do algoritmo é determinada pela condição do *loop while* externo. O tradutor extrai essa condição de parada em um filtro que fica responsável por checa-la, no caso das aplicações de mineração de dados utilizadas nessa dissertação, verificar se o resultado convergiu.

As operações de Map-Reduce são então identificadas e cada uma delas é extraída para entrar num filtro.

Para cada operação de Map-Reduce identificada, o tradutor gera um filtro contendo a computação realizada pelo map-reduce, sendo que a comunicação é realizada de acordo com a dependência de dados existente.

$$L_x \leftarrow \text{map } (\text{ reduce } (f_r)\ L_a)\ L_b \tag{3.1}$$

É importante falar que o apesar de apresentar as operações de Map-Reduce dessa forma, o tradutor identifica o esqueleto das funções em C, e não em linguagem funcional. Por exemplo, o Map-Reduce da equação 3.1 é identificado em linguagem imperativa como sendo o algoritmo representado na figura 3.5 .

```
int Lx[NB]; int Lb[NB] = ...; int La[NA] = ...;
for (ib = 0; ib < NB; ib++) {
  Lx[ib] = InitialValue(Lb[ib]);
  for (ia = 0; ia < NA; ia++) {
    Lx[ib] = fr(Lx[ib], La[ia]);
  }
}
```

Figura 3.5: Operação de Map-reduce escrita em C, representando a equação 3.1.

O foco do tradutor é a geração de código que executa em paralelo. Não há preocupação com a otimização do código gerado nem com possíveis gargalos gerados na leitura ou na escrita de dados, se desejar o programador pode assumir que o dado de entrada chega através de um *stream* e o dado de saída deve ser escrito num *stream*, focando a compilação no trabalho propriamente dito.

Ao assumir que a memória é infinita, o usuário espera que o dado esteja disponível sempre que o acesso for realizado. Em sistemas reais, sabe-se que isso não é verdade, mas o tradutor também assume o mesmo que o usuário, deixando problemas de falta de memória, ou acesso incorreto para serem detectados posteriormente ao processo de geração do código dos filtros.

Para facilitar a avaliação das estratégias propostas nessa dissertação, optou-se por limitar o escopo do uso do *C*, permitindo foco nas transformações. O código deve

estar descrito em somente uma função. Nenhum chamada de função externa é avaliada e uma postura conservadora é assumida, ou seja, uma chamada de função altera todas as variáveis que são parâmetros. Não existe alocação dinâmica dentro do código, não há uso de *goto* ou *switch/case* no código. O uso desse escopo limitado de C diminui a quantidade de análises a serem realizadas permitindo o foco apenas nas estratégias apresentadas nessa dissertação, mas alguns algoritmos não podem ser paralelizados pelo tradutor, por exemplo, algoritmos que utilizam recursividade ou com orientação a objeto.

O programador que decidir utilizar o tradutor para gerar código automaticamente deve estar ciente de que se o código não tiver essas características ele não será dividido em filtros, ou os filtros gerados poderão possuir a obrigatoriedade de uma única instância de execução.

3.2.1 Informações técnicas

O tradutor transforma código sequencial escrito em C em programas escritos em C que utilizam a biblioteca do Anthill. O compilador original Cetus contêm 74000 linhas de código Java comentado, e as modificações implementadas contém cerca de 6400 linhas de código, divididas em: conversão para SSA (1700 linhas); análise estática para descoberta de padrões map-reduce (3400 linhas); impressão dos códigos dos filtros (1300 linhas).

3.2.2 Formato intermediário do algoritmo

Antes de mostrar o processo de compilação em si, é necessário descrever o formato intermediário utilizado pelo tradutor. Esse formato está descrito na documentação do Cetus [Lee et al., 2003, Johnson et al., 2004].

O tradutor armazena o código num formato intermediário representado através de uma *Abstract Syntax Tree* (AST), que é uma árvore construída para permitir que o código original seja gerado da mesma forma que ele foi escrito, ou seja, ela representa o fluxo do programa de forma hierárquica. Cada um dos nós da árvore representa uma característica do código. Por exemplo, os filhos de um nó representam o código dos fluxos que estão inseridos na sua hierarquia (num *loop*, os comandos que estão dentro dele são representados como filhos, numa atribuição de variável, cada variável é representada como filhas do nó atribuição, e assim por diante).

Tipos de nó existentes no formato intermediário:

Programa: é a raiz da árvore, o tradutor é construído para tratar de somente um "passo"de compilação.

Unidade de Tradução: representa cada um dos arquivos que estão sendo analisados.

Declarações: declarações de funções, variáveis ou classes.

Expressões: atribuições, chamadas de funções, ou seja, as computações.

Comandos: representam cada uma das linhas do código representando os comandos da linguagem (*while*, por exemplo), são usados para armazenar as declarações e expressões, montando a estrutura do código.

Adjetivos: representam os especificadores da linguagem, por exemplo, *static long int*.

O código intermediário é transformado no formato *Static Single Assignment* (SSA) [Cytron et al., 1989] para facilitar a construção da cadeia de uso e definição, que define os pontos onde uma variável é declarada e onde ela é utilizada. Nesse formato cada nova definição de uma variável é numerada, e quando o fluxo do programa permitir mais de um fluxo possível para uma variável a ser utilizada, uma função ϕ é inserida definindo uma nova variável. Dessa forma, sabe-se o real posicionamento onde as funções de comunicação devem ser inseridas.

3.3 Tarefas do tradutor

O processo de compilação possui diversos algoritmos que realizam a análise dos dados e as modificações no código original. O algoritmo 2 mostra os passos que são realizados pelo tradutor, as informações extraídas pelo tradutor (CFG, lista de *loops, defs, uses*) são combinadas para transformar o código para *Static Single Assignment* [Cytron et al., 1989] ou SSA e posteriormente para gerar os filtros e comunicação.

A análise léxica do programa de entrada é realizada pelo Antlr [Parr & Quong, 1995] e o código é colocado no formato interno de representação, uma árvore AST. A partir dessa árvore se extrai do código o CFG, contendo o fluxo do programa. O código é, então, modificado para o formato SSA.

Cada umas das funções executadas pelo tradutor realiza transformações no código. Para exemplificar essas transformações, será utilizado o algoritmo 3, que é o algoritmo que faz o cálculo de divisores frequentes apresentado anteriormente.

Algoritmo 2: Processo de tradução

$cfg = GenerateCFG(Program)$;
$RemoveUnreachableCode(Program)$;
$pragma\_list = GetPragmaList(Program)$;
$GenerateSSA(program, cfg, pragma\_list)$;
$statements = GetStatementList(program, cfg)$;
$external\_while\_loop = GetExternalWhileLoop(program, cfg)$;
$loops = GetLoopList(statments, cfg)$;
$filters = ExtractMapReduce(statments, cfg, loops, external\_while\_loop)$;
$GetFilterInformation(filters, statements, cfg)$;
$FillDefUseChain(filters)$;
$InsertCommunication(filters)$;

3.3.1 Grafo de Fluxo de Controle

A figura 3.6 mostra o grafo de Fluxo de controle do Algoritmo de Cálculo de Divisores Frequentes. Um grafo de fluxo de controle mostra os blocos básicos do código e os caminhos possíveis que o fluxo do programa pode seguir.

Essa estrutura de dados é importante para a geração de informações sobre as dependências de dados entre os filtros que serão gerados.

3.3.2 Remoção de código inalcançavel

Essa função realiza a remoção de todos os blocos presentes no Grafo de Fluxo de Controle que são alcançados. Essa é a única otimização presente nesse compilador, e serve para evitar a geração de filtros que não realizam qualquer tarefa. No caso do CFG mostrado na figura 3.6 não há nós inalcançaveis e nehuma remoção é feita.

3.3.3 Diretivas manuais

A lista de *pragmas* ou diretivas é extraída pelo tradutor. Essas diretivas são utilizadas para marcar no código informações que o usuário do tradutor julgar relevante informar ao tradutor, por existir uma informação que não foi corretamente identificada ou para facilitar o trabalho de análise. Essas diretivas são respeitadas e assume-se que estão sempre corretas, são elas:

- `#pragma ANTHILL`

  Avisa ao tradutor que o código possui diretivas definidas pelo programador.

- `#pragma DISTRIBUTED <NOME DA VARIÁVEL>`

Algoritmo 3: Algoritmo de Cálculo dos Divisores Frequentes

```
int main(int argc, char * argv[]) {
#pragma ANTHILL
    char atributeFileName [100];
    char transactionFileName [100];
    int support, value, atributes, i, j, transactions;
    FILE * Arq;
    int atribute[MAX];
    int transaction[MAX];
    int count[MAX];
#pragma READ BEGIN
    {
        { ... }
        atribute[]=readFile(atributeFileName);
        { ... }
        transaction[]=readFile(transactionFileName);
       { ... }
    }
    for (j=0;j<atributes;j++){
       count[j]=0;
    }
    for (i =0;i<transactions;i++){
       for (j=0;j<atributes;j++){
          if(transaction[i]%atribute[j]==0){
             count[j]++;
          }
       }
    }
    for (j=0;j<atributes;j++){
       if (count[j]>=support){
          printf("Atribute: %d is frequent (%d)\n", atribute[j],count[j]);
       }
    }
    return 0;
}
```

Identifica a variável que está distribuída em várias máquinas no *cluster*, em geral essa variável depende de um arquivo de entrada que é diferente para cada uma das máquinas.

- `#pragma INDUCTION <NOME DA VARIÁVEL>`

  Identifica uma variável de indução.

- `#pragma DEPENDENCE <NOME DA VARIÁVEL1> -> <NOME DA VARIÁVEL2>`

  Identifica que a variável 1 depende da variável 2. Essa informação é importante principalmente quando se deseja mostrar dependências que não podem ser identificadas pelos algoritmos utilizados nesse trabalho.

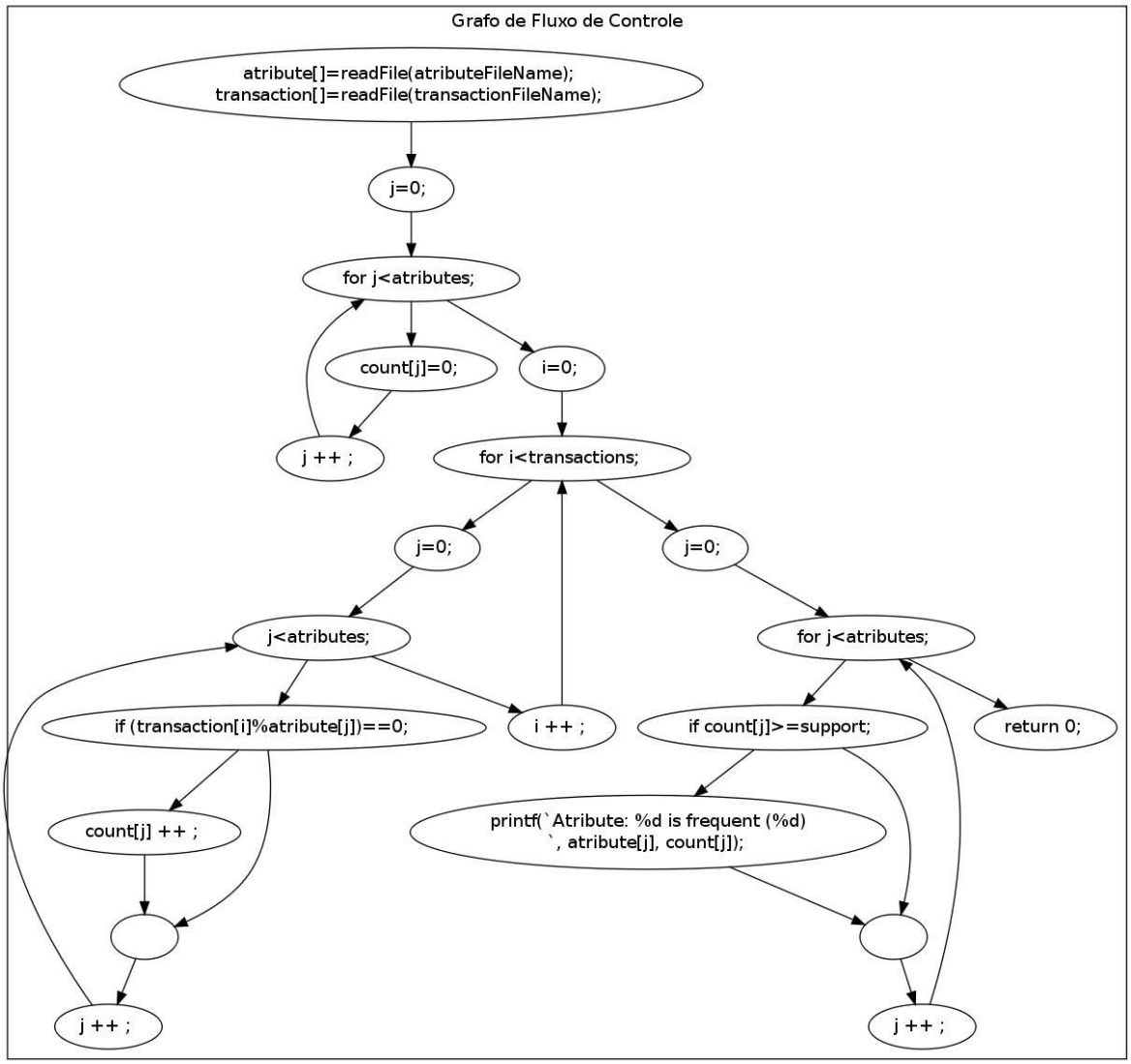


Figura 3.6: CFG do algoritmo de Cálculo de Divisores Frequentes

- `#pragma REDUCTION <NOME DA VARIÁVEL> <OPERAÇÃO>`

  Identifica um *loop* de redução.

3.3.4 Geração do código no Formato SSA

Para cada ponto do Grafo de Fluxo de Controle onde uma variável pode receber o seu valor de duas ou mais definições diferentes, uma função ϕ é criada contendo as definições possíveis. Dessa forma é possível saber o local onde comunicações devem ser inseridas.

O algoritmo 4 mostra o Algoritmo de Cálculo de Divisores Frequentes no formato SSA.

Algoritmo 4: Algoritmo de Cálculo dos Divisores Frequentes no formato SSA

```
int main(int argc, char * argv[]) {
#pragma ANTHILL
    char atributeFileName[100];
    char transactionFileName[100];
    int support, value, atributes, i, j, transactions;
    int atribute[2048];
    int transaction[2048];
    int count[2048];
#pragma READ BEGIN
    {
        { ... }
        atribute#1[0]=readFile(atributeFileName#0);
        { ... }
        transaction#1[0]=readFile(transactionFileName#0);
        { ... }
    }
    for (j#1=0; j#1<atributes#0; j#1 ++ ) {
        count#0[j#1]=0;
    }
#pragma PHI j#2 <- phi ( j#1 , j#3 ) ;
#pragma PHI count#2 <- phi ( count#0 , count#1 ) ;
    for (i#1=0; i#1<transactions#0; i#1 ++ ) {
        for (j#3=0; j#3<atributes#0; j#3 ++ ) {
            if ((transaction#1[i#1]%atribute#1[j#3])==0) {
                count#1[j#3] ++ ;
            }
        }
    }
    for (j#4=0; j#4<atributes#0; j#4 ++ ) {
        if (count#2[j#4]>=support#0) {
            printf("Atribute: %d is frequent (%d)\n", atribute#1[j#4], count
                #1[j#4]);
        }
    }
    return 0;
}
```

3.3.5 Extraindo os Loops

O *loop* mais externo do código, representado pelo *loop while* no *loop* canônico, é extraído, ele contém a condição de parada do algoritmo.

As operações de Map-Reduce são então identificadas e cada uma delas é extraída para entrar num filtro. Caso um *loop* não possa ser identificado como estando no padrão de operação de Map-Reduce apresentado na figura 3.5, um filtro é criado e

uma mensagem é mostrada ao usuário avisando sobre a obrigatoriedade de execução de somente 1 instância desse filtro. Essa informação também é inserida no arquivo de configuração XML do layout de filtros.

Para encontrar quais os dados devem ser comunicados, utiliza-se a análise padrão de *reaching definition* [Aho et al., 2006]. Para cada variável a árvore de uso-definição é criada e basta descobrir a última definição antes da utilização, o que é simples visto que o código encontra-se no formato SSA. Elas são utilizadas para definir quais serão as comunicações realizadas. O tradutor sempre adota uma postura conservadora, caso não descubra que uma comunicação é desnecessária, essa comunicação é sempre realizada. Esse é o preço que se paga pelo compromisso de gerar *código correto* × *código eficiente*. Quanto mais genérico é um compilador, mais casos de borda necessitam de tratamento especial.

Outra informação extraída pelo tradutor é o grafo de dependência de dados. Esse grafo possui a relação entre uso e definição das variáveis, mostrando quais devem estar disponíveis para a atualização de outras variáveis. Dentro de um *loop*, cada variável atualizada pode possuir um caráter específico, por exemplo, se uma variável é redefinida em cada iteração, provavelmente é um variável de uso temporário. Os tipos de dependências de dados possíveis entre as variáveis podem viabilizar a paralelização de um *loop*:

- *flow dependence*: É a dependência de leitura depois de escrita, ou seja numa iteração do *loop* a variável é atualizada e em outra ela é lida. Em geral esse tipo de dependência obriga a execução das iterações do *loop* de forma ordenada. Se uma dependência desse tipo é descoberta, o *loop* é considerado não paralelizável.

- *anti dependence*: É a dependência de escrita depois de leitura. A privatização de variáveis pode ser utilizada para garantir a leitura correta do valor.

- *output dependence*: É a dependência de escrita depois de escrita, que ocorre quando várias iterações escrevem numa mesma posição de memória. Normalmente ocorre na atualização de um estado do programa ou no armazenamento de um valor temporário. Deve-se tomar cuidado sobre qual o último valor que deve ser escrito.

A privatização de uma variável só é possível se essa variável é escrita antes de ser lida dentro da mesma iteração do *loop*. Isso deve ser verdade para todas as leituras feitas. No caso de uma variável estar disponível fora do *loop* deve-se tomar cuidado para copiar o valor correto para a versão não privatizada da variável.

As relações de dependências entre as variáveis trazem informações importantes sobre os particionamentos possíveis do código. Em geral as variáveis de redução são as variáveis que carregam o resultado da computação realizada, ou seja, o objetivo de um *loop* é calcular o valor final dessas variáveis. Dessa forma se pode extrair as dependências entre essas variáveis para formar um grafo relacionando-as. Esse grafo, além de mostrar a relação entre as variáveis, também demonstra o nível de comunicação necessário se o código gerado para os filtros separasse a definição do uso da variável.

Com o grafo de dependências de dados e controle em mãos, o tradutor pode extrair as informações sobre as variáveis de trabalho. As variáveis de trabalho são aquelas que possuem a primeira geração a partir dos dados de entrada e as gerações seguintes são feitas no *loop* mais externo (*while*). Normalmente a geração N é dependente dos dados da geração N-1, além dos dados de entrada. Essa dependência é traduzida com uma realimentação no grafo de execução do algoritmo, que nesse caso, define o *pipeline* de filtros do Anthill. Assume-se que o código responsável pela geração de cada uma das variáveis do grafo é parte de um filtro e as arestas de dependências representam a comunicação entre eles.

Em alguns casos os dados lidos podem estar distribuídos em várias máquinas e o Anthill deve executar instâncias de um mesmo filtro nessas máquinas (cópias transparentes). O nosso tradutor assume que os arquivos lidos em cada uma das cópias transparentes possuem o mesmo nome, a não ser que o programador ou o ambiente passe nomes de arquivos diferentes como parâmetro para cada instância. Se instâncias de um mesmo filtro precisarem de comunicação entre si, assume-se que o sistema não realizará cópia transparente desse filtro, mesmo que isso crie uma grande limitação de tamanho de memória (assume-se que toda a memória necessária estará disponível para a execução do algoritmo) ou capacidade de processamento. A comunicação entre instâncias de um mesmo filtro é possível, porém não é recomendada.

Assume-se somente 2 tipos de recepção de dados, um filtro pode receber dados de todas as instâncias de um filtro anterior ou receber dados de somente uma instância de um filtro anterior. Essa limitação é importante pois não se sabe, a priori, a quantidade de instâncias que o Anthill pode gerar, além disso não há controle do número de mensagens trocadas entre os filtros, uma instância de um filtro não possui qualquer informação da outra instância.

3.3.6 Definindo as comunicações

Existem duas possíveis formas de comunicação entre filtros, a primeira utilizando uma combinação de *broadcast* e *round-robin* e a segunda, uma otimização utilizando *labeled-*

stream. Por padrão, cada operação de Map Reduce é traduzida para a primeira opção, apenas se uma condição especial é encontrada, utiliza-se a segunda opção.

Uma operação de Map Reduce como aquela mostrada na equação 3.1 é traduzida para um filtro que vai receber os dados L_a via *broadcast*, e os dados L_b via *round-robin*. O código gerado está apresentado no algoritmo 5. As funções RcvBc e RcvRr representam, respectivamente as operações de *broadcast* e *round-robin*.

Algoritmo 5: – **Estratégia padrão de paralelização produzida**:

1: $L_x \leftarrow [\iota, \ldots, \iota]$
2: $L_a \leftarrow$ RcvBc()
3: **while** $b \leftarrow$ RcvRr() **do**
4: $\quad L_x[b] \leftarrow$ reduce f_r L_a
5: **end while**

A estratégia padrão distribui o trabalho realizado sobre L_b igualmente entre as diversas instâncias do filtro. Entretanto, é possível realizar uma técnica de paralelização mais agressiva quando a operação de redução $\oplus$ é um operador associativo que segue o padrão como o apresentado na equação 3.2.

$$f_r = \lambda a\ .\lambda(b, a_{acc})\ .\ \text{if } a \subseteq b \text{ then } (b \setminus a) \oplus a_{acc} \text{ else } a_{acc} \tag{3.2}$$

Não é necessário enviar toda a estrutura L_a (conforme visto na equação 3.1) via *broadcast* para todas as instâncias do filtro responsável. Ao invés disso, L_a pode ser dividido utilizando uma função de hash implementando a comunicação através de *label-stream*. O filtro produzido contém apenas a computação do "then"da função de redução, ou seja: $\lambda a\ .\ \lambda(b, a_{acc})\ .\ (b \setminus a) \oplus a_{acc}$.

O algoritmo 6 mostra o código gerado para essa construção, onde RcvLs é a função de leitura de dados utilizando *labeled-stream*. Para cada tupla (b, a), onde a e b são parâmetros da função de redução da equação 3.2, cria-se uma mensagem com 2 campos, um label formado por b, e um valor, formado pela diferença entre os conjuntos $b \setminus a$.

Algoritmo 6: – **Código paralelo para a otimização utilizando *label-stream***:

1: $L_x \leftarrow [\iota, \ldots, \iota]$
2: **while** (`label, value`) $\leftarrow$ RcvLs() **do**
3: $\quad L_x$[`label`] $\leftarrow$ L_x[`label`] $\oplus$ `value`
4: **end while**

A figura 3.7 compara o código que a otimização de *label-stream* produz. Figura 3.7 (a), mostra o *loop* map-reduce loop em C, contendo o padrão mostrado na equação 3.2. Figura 3.7 (b) mostra o código paralelo gerado sem a otimização e a figura 3.7 (c) mostra o código com a otimização aplicada.

a) Original, código sequencial

```
int Lb[NB] = ...;
int La[NA] = ...;
int Lx[NB] ;
for (ib = 0; ib < NB; ib++) {
   Lx[ib] = InitialValue (Lb[ib]) ;
   for (ia = 0; ia < NA; ia++) {
      if (Subset (La[ia], Lb[ib])) {
         Lx[ib] = fr (Lx[ib], Dif (La[ia], Lb[ib])) ;
      }
   }
}
```

b) Código paralelo sem a otimização com label-stream

```
int *Lx = InitArray() ;
int *La = RcvBc() ;
RRData* bdt;
while (bdt = RcvRr()) {
   for (ia = 0; ia < NA; ia++) {
      if (Subset (La[ia], bdt)) {
         Lx[bdt] = fr (Lx[bdt], Dif (La[ia], Lb[ib])) ;
      }
   }
}
```

c) Código paralelo com a otimização usando label-stream

```
LSData* adt;
while (adt = RcvLs()) {
   Lx[adt->label] = fr (Lx[adt->label], adt->value) ;
}
```

Figura 3.7: Comparação entre o código gerado com e sem a otimização para utilizar *label-stream*.

O próximo passo é a definição da função de hash h, que é criada em tempo de compilação. Valores do conjunto L_a (equação 3.1) são mapeados em filtros. Ou seja, dado um $a \in L_a$, onde $\exists b \subseteq a, b \in L_b$, temos $h(b) = s$, é o filtro onde a mensagem a deve ser entregada.

O domínio de iteração dos *loops* é checado e a função de hash é definida para encaminhar as variáveis de acordo com o domínio.

O loop que testa se uma transação é divisível por um atributo no algoritmo de Cálculo de Divisores Frequentes, mostrado no algoritmo 7 define uma comunicação do tipo *label-stream* conforme os padrões apresentados anteriormente. Nesse caso, as transações são enviadas através de *round-robin* e as transações utilizam o *label-stream*.

Depois de retirar essas informações, sobra, no algoritmo original, a parte de leitura de dados e a parte de escrita de dados. Assume-se que o programador que deseja que filtros sejam gerados para leitura e escrita de dados, tenha inserido diretivas mostrando ao tradutor onde são feitas essas partes. Cada uma delas é extraída do código original e inserida num novo filtro. O filtro de leitura realiza a distribuição dos dados lidos através de *round-robin*, e é instanciado em todos os lugares onde houver dado para ser lido. O filtro de escrita é instanciado somente uma vez, para evitar controle de dados de saída.

Algoritmo 7: Loop do algoritmo de Cálculo dos Divisores Frequentes

```
for (i =0;i<transactions;i++){
    for (j=0;j<atributes;j++){
        if(transaction[i]%atribute[j]==0){
            count[j]++;
        }
    }
}
```

O restante do código original contém as informações que são inseridas no *Work*. Essas informações são os parâmetros de entrada do programa e, em geral, contem as informações para a leitura dos dados de entrada (como nome de arquivos).

3.3.7 Sumário

Nesse capítulo foram apresentadas duas paralelizações distintas para o problema de Divisores Frequentes e como a escolha de comunicações a serem realizadas influência no desempenho da aplicação. Além disso as estratégias de compilação utilizadas foram descritas. Conforme apresentado, padrões Map-Reduce são identificados no código que está no formato de Loop Canônico e esses padrões são traduzidos escolhendo-se a estratégia de comunicação desejada.

Capítulo 4

Aplicações Avaliadas

Como foi apresentado anteriormente, várias das aplicações seguem o modelo do *loop* canônico. Para realizar os experimentos dessa dissertação, optou-se pelo uso de três aplicações de mineração de dados, K-means, Frequent itemsets e K-nearest neighbors, que podem demonstrar o uso das propostas apresentadas no capítulo 3. Nesse capítulo serão apresentas as aplicações escolhidas, qual o comportamento do tradutor ao lidar com essas aplicações e quais filtros foram gerados. Para facilitar a compreensão do processo de compilação, serão apresentados apenas pseudo-códigos, ao invés do código completo em *C*.

4.1 K-means

O Algoritmo K-means [Macqueen, 1967, Steinhaus, 1956] é utilizado para dividir um conjunto de elementos em k sub-conjuntos de forma que cada um deles contenha elementos com características similares. Assume-se que os elementos estão num espaço vetorial onde cada dimensão representa uma característica, são realizadas iterações movendo os pontos para os centros mais próximos, e depois calculando os novos centros até que não haja mais mudança de pontos entre iterações.

Nesse exemplo assume-se que os elementos são pontos num espaço euclidiano. A entrada é uma lista L_p de pontos que devem ser agrupados em clusters, além disso há uma lista L_c de K *centroids*. Os centroids são pontos no espaço euclidiano. Um ponto $p \in L_p$, próximo a $c \in L_c$, é parte do cluster c.

Assume-se que a posição inicial dos centroids é escolhida aleatoriamente. Durante a execução do algoritmo, cada centro vai convergir para a posição que tende a

maximizar o número de pontos que fazem parte desse cluster. O algoritmo termina quando não há mais mudança na posição dos pontos ou quando um número máximo de iterações é alcançado. O algoritmo 8 mostra o K-means. Há 2 operações de map-reduce, sendo a primeira, da linha 6, responsável por mapear pontos $p \in L_p$ em tuplas (p, c), onde c é centroid de qual p está mais próximo. A segunda operação varre a lista de centroids L_x e, para cada centroid c, a redução varre a lista L_t. Dado o elemento $(p, c') \in L_t$, sempre que $c = c'$, o valor p é utilizado para atualizar a posição c.

Algoritmo 8: – **K-means**:

1: $L_p \leftarrow$ list of points
2: $L_c \leftarrow$ initial centroids
3: $L_x \leftarrow \emptyset$
4: **while** $L_x \neq L_c$ **do**
5: $\quad L_x \leftarrow L_c$
6: $\quad L_t \leftarrow$ map(λp . reduce (minDistanceFromPointP p) L_x) L_p
7: $\quad L_c \leftarrow$ map(λc . reduce (updateCentroidIfEqualToC c) L_t) L_x
8: **end while**
9: **return** L_c

A figura 4.1 mostra os filtros que o tradutor produz para o algoritmo 8. Nesse e nos próximos algoritmos utilizaremos a notação da figura 4.1 (b) para descrever os filtros Anthill gerados. São criados 2 tipos de filtros, chamados de F_{sep} e F_{upd}. F_{sep} separa pontos de acordo com o centroid mais próximo, dado um conjunto de N pontos $\{p_1, \ldots, p_N\}$ e um conjunto de K centroids $\{c_1, \ldots, c_K\}$, F_{sep} produz um conjunto de N tuplas $\{(p_1, c_i), \ldots, (p_N, c_j)\}$, onde cada ponto é mapeado para o centroid mais próximo. O Filtro F_{upd} atualiza a posição dos centroids dado as posições dos pontos compõe o cluster.

Ao implementar a primeira operação de map-reduce, na linha 6, dado a natureza do algoritmo, pode-se dividir igualmente os pontos nas instâncias do filtro F_{sep}, mas as instâncias deverão receber todo o conjunto de centroids. Os pontos são enviados através de round-robin e os centroids enviados utilizando-se broadcast.

Pode-se dividir os centroids igualmente os filtros F_{upd}, mas o filtro que for o responsável por atualizar o centroid c deverá ter acesso a todos os pontos desse centro c. Portanto, a forma como os dados são enviados para os filtros F_{upd} depende de informações produzidas em tempo de execução, ou seja, o centroid no qual um ponto for assinalado determina o filtro F_{upd} para onde esse ponto deve ser enviado. Este é o caso de uma comunicação utilizando *labeled stream*, onde a mensagem é o ponto e seu label é o centroid para onde o ponto é assinalado.

Existe apenas um filtro f_{upd}, instância de F_{upd}, responsável por atualizar o centroid c. Se um ponto p está no centro próximo a c e existe uma tupla $(p, c) \in L_t$, então p será enviado somente para f_{upd}.

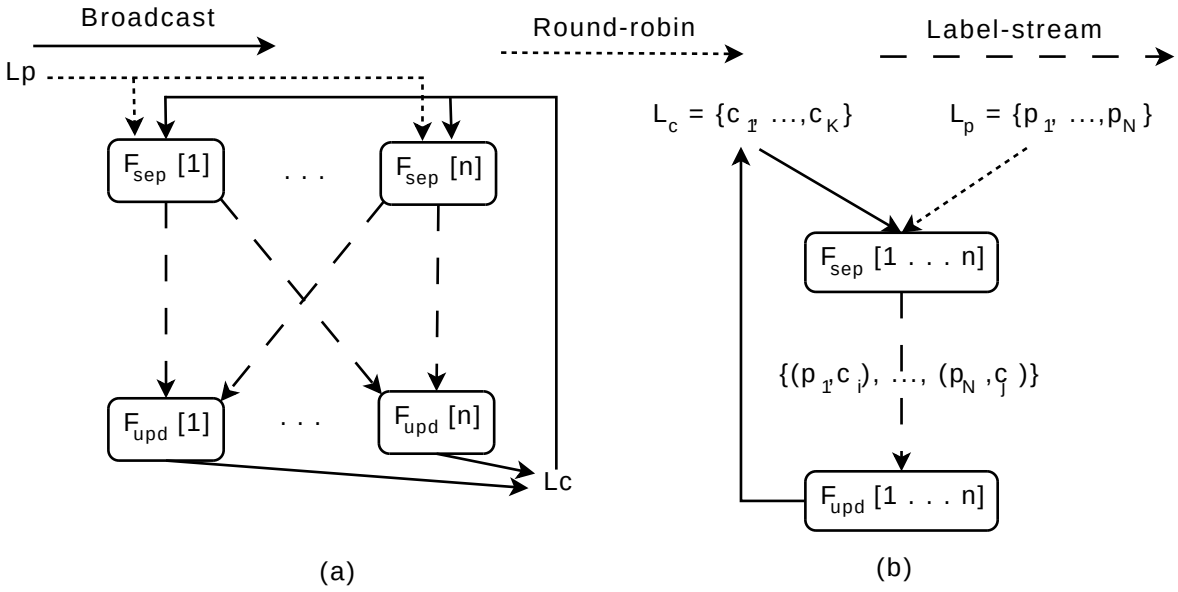


Figura 4.1: (a) K-means representado como uma coleção de filtros. (b) Visão simplificada.

4.2 Frequent Itemsets

O algoritmo Frequent itemsets [Agrawal et al., 1993] é utilizado para saber, dentro de um conjunto de itens com características discretizadas, quais são as combinações dessas características que são frequentes, ou seja, que possuem a contagem maior que um parâmetro de entrada. As características também são chamadas de *itemsets*.

Nessa dissertação utilizou-se a versão conhecida como *apriori*. A entrada do algoritmo é um número inteiro σ, chamado suporte, e um conjunto de transações $L_t = \{t_1, t_2, \ldots t_n\}$, sendo que cada transação é um conjunto de atributos da seguinte forma: $t_i = \{a_1, a_2, \ldots, a_m\}$. Transações podem ter cardinalidades diferentes. O objetivo do algoritmo é descobrir quais sub-conjuntos dos atributos são frequentes, ou seja, quais deles ocorrem em mais de σ transações. Um descrição dessa técnica pode ser vista no algoritmo 9.

O loop canônico é composto de três sequencias de reduções. A primeira, na linha 10, atribui candidatos as transações quando essas ocorrem, produzindo uma lista L_n de tuplas (c, t). A segunda, na linha 11, conta quantas vezes cada candidato ocorre,

varrendo a lista de candidatos, para cada candidato c, varre-se L_n, e soma as tuplas que casam com (c, t), para um $t \in L_t$. Os candidatos frequentes também são separados. A terceira redução produz um nova lista de candidatos a serem verificados. Novos candidatos possuem a cardinalidade N e são produzidos pareando candidatos frequentes de cardinalidade $N - 1$. Alguma otimizações óbvias são omitidas para simplificar a apresentação.

Algoritmo 9: – **Frequent Itemsets**:

1: $\sigma \leftarrow$ support value
2: $L_t \leftarrow$ list of transactions
3: $L_a \leftarrow$ attributes that occur more than σ times among transactions
4: $L'_c \leftarrow L_a$
5: $L_{res} \leftarrow \emptyset$
6: $L_c \leftarrow \emptyset$
7: **while** $L_c \neq L'_c$ **do**
8: $\quad L_{res} \leftarrow L_{res} +\!\!+ L'_c$
9: $\quad L_c \leftarrow L'_c$
10: $\quad L_n \leftarrow$ map (λt . reduce (getCandidatesThatAreInT t) L_c) L_t
11: $\quad L_f \leftarrow$ removeUndefs (map (λc . if (reduce (countNumberOfC c) L_n) $>$ σ then c else $\perp$) L_c)
12: $\quad L'_c \leftarrow$ map (λc . reduce (λa . $a \cup c$) L_a)) L_f
13: **end while**
14: **return** L_{res}

O lado esquerdo da figura 4.2 mostra os filtros que o tradutor produziu para o algoritmo 9. Na primeira fase cada transação é processada independente das anteriores, mas 1 transação precisa de todos os candidatos, ou seja, candidatos são enviados para instâncias do filtro F_{cnt} através de broadcast, enquanto transações chegam através de round-robin. Para cada transação que chega, um filtro produz o par (c, t), contendo o candidato c que é frequente na transação t. Esse dado é enviado através de label-stream para o filtro F_{sup}, que soma todas as ocorrências de um determinado candidato. O candidato é enviado, se e somente se a sua contagem é superior ao valor de suporte. O label utilizado é o candidato.

4.3 K-nearest neighbors

O algoritmo K-nearest neighbors ou *k-vizinhos mais próximos* (*knn*) [Witten & Frank, 2002] é uma técnica que classifica cada um dos elementos de um conjunto L_q de *consultas* utilizando os K elementos mais próximos de um conjunto L_t de treinamento. Esse método está descrito no algoritmo 10, que é

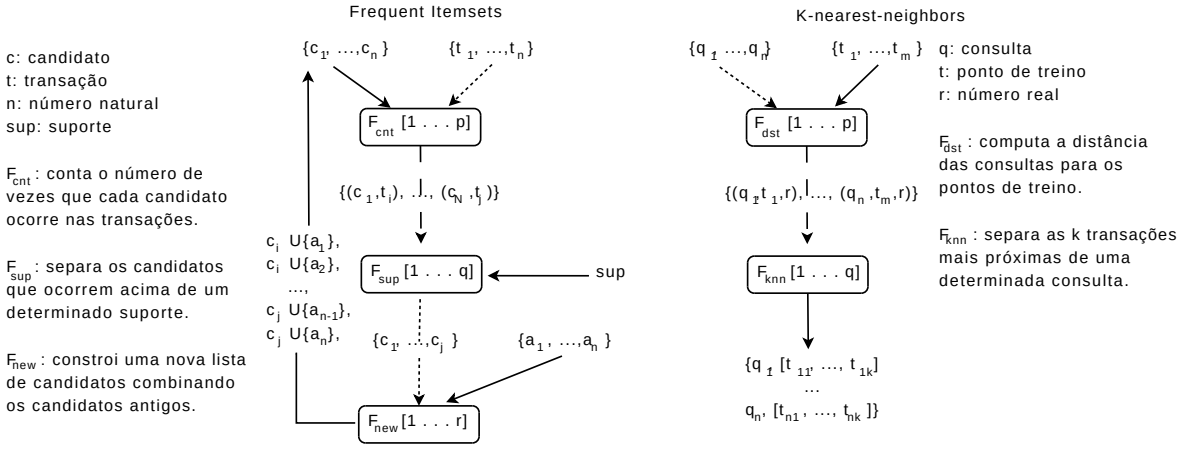


Figura 4.2: (Esquerda) Filtros do Frequent Itemsets. (Direita) Filtros do K-nearest neighbors.

uma versão *degenerada do loop canônico*, ou seja, ele contém apenas 2 operações de map-reduce, mas essas operações são executadas apenas 1 vez. A primeira operação, na linha 3, contém apenas maps; para cada consulta q ela produz uma lista $|L_t|$ de tuplas $(q, t, r), t \in L_t$, onde r é a distância entre q e t. A segunda operação, na linha 4, encontra os K pontos de treinamento próximos a q.

Algoritmo 10: – **K Nearest Neighbors**:

1: $L_q \leftarrow$ List of queries
2: $L_t \leftarrow$ Training set
3: $L_d \leftarrow$ map $(\lambda q$. map $(\lambda t$. $(q, t,$ distance(q, t))) $L_t)$ L_q
4: $L_k \leftarrow$ map $(\lambda q$. reduce (findTheKClosestTraningPointsToQ $q)$ $L_t)$ L_q
5: **return** L_k

Os filtros produzidos para essa aplicação estão apresentados no lado direito da figura 4.2. O tradutor gera 2 tipos de filtros, F_{dst}, que computa a distância de cada consulta para cada ponto de treino, e F_{knn}, que encontra os K pontos de treino mais próximos a cada consulta, dada as distâncias computadas anteriormente. A comunicação entre F_{dst} e F_{knn} é feita utilizando labeled streams, pois podemos encontrar pontos que estão mais próximos a cada consulta de forma independente das outras consultas. Cada mensagem (q, t, r) tem o label como sendo a consulta q, contém um ponto de treino t e a distância r entre t e q.

4.3.1 Sumário

Nesse capítulo foram apresentadas as aplicações utilizadas para demonstrar a qualidade dos algoritmos de paralelização automática do tradutor proposto e como essas aplicações foram paralelizadas utilizando as estratégias propostas.

Capítulo 5

Experimentos Realizados

Para avaliar os filtros que foram gerados através do tradutor, foram executados experimentos variando o número de instância de filtros, medindo o tempo de execução de cada um dos filtros. O tempo de execução total do algoritmo teve os tempos de leitura e escrita de dados excluídos, pois o objetivo foi avaliar a eficiência dos filtros que efetivamente realizam a computação. Além disso as versões geradas automaticamente foram comparadas com versões dos mesmo algoritmos implementadas manualmente por programadores experientes.

Os experimentos foram executados num *cluster* contendo 36 computadores, sendo cada um composto por um processador AMD Athlon 64 3200+ (2GHz), com cache de 512KB, 2GB de memória RAM cada e discos SATA (ATA 150) de 160GB. Os computadores foram ligados através de rede *Gigabit Ethernet*. Os equipamentos possuíam dedicação exclusiva para a execução dos experimentos. O sistema operacional utilizado foi o *Debian GNU/Linux* 4.0 com *kernel* na versão 2.6.18-6. A versão do Anthill utilizada foi a 3.1.

Optou-se por executar os experimentos variando o número de instâncias dos filtros apresentados nas sessões do capítulo 4, e o tamanho da base de dados. Para cada um dos algoritmos serão apresentadas as configurações dos experimentos realizados.

Foram executados dois grandes conjuntos de experimentos, no conjunto apresentado na seção 5.1, foi realizada a avaliação de speed-up e scale-up dos algoritmos apresentados medindo o tempo de execução variando o número de instâncias dos filtros. No segundo conjunto, apresentado na seção 5.2, o tempo de execução dos algoritmos compilado foi comparado com uma versão desenvolvida por um programador sem a ajuda do tradutor apresentado nessa dissertação.

5.1 Avaliação do tradutor

Nessa seção são apresentados os gráficos de *speed-up* e *scale-up* para todos os algoritmos. Os gráficos de *speed-up* foram apresentados para cada uma das bases de dados escolhidas e os gráficos de *scale-up* foram apresentados iniciando o número de filtros em vários valores diferentes. Além disso, são apresentadas tabelas para mostrar mais claramente os valores de *speed-up* e *scale-up* nos casos onde se fez necessário.

5.1.1 Frequent itemsets

O tradutor gera os filtros de leitura e escrita de dados, e o filtro raiz, que não precisam de ser executados com mais de uma instância [1]. Os filtros foram apresentados na seção 4.2, e foram instanciados várias vezes, para realizar a avaliação da aplicação.

Filtro/Recurso:	Variação:
Filtro F_{cnt}	1;2;4;8;12 e 16 instâncias
Filtro F_{sup}	1;2;4;8;12 e 16 instâncias
Base de dados	2,5; 5 e 10 milhões de pontos

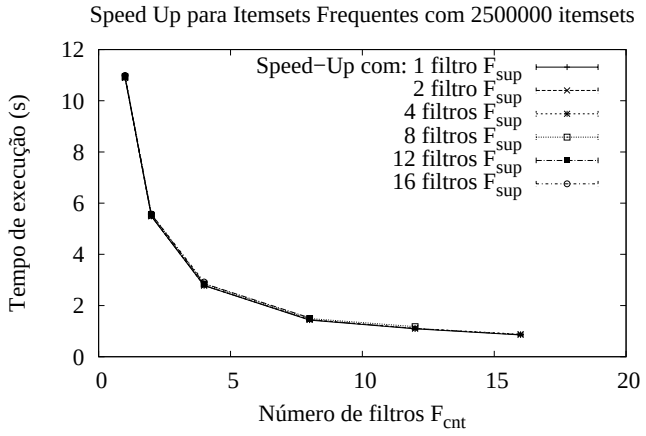
Tabela 5.1: Variações para execução do algoritmo Frequent itemsets

Foram realizados experimentos com todas as combinações das variações apresentadas na tabela 5.1. Cada um dos pontos da base é composto de 20 características possíveis, sendo que para escolhê-las, foi utilizada uma distribuição aleatória.

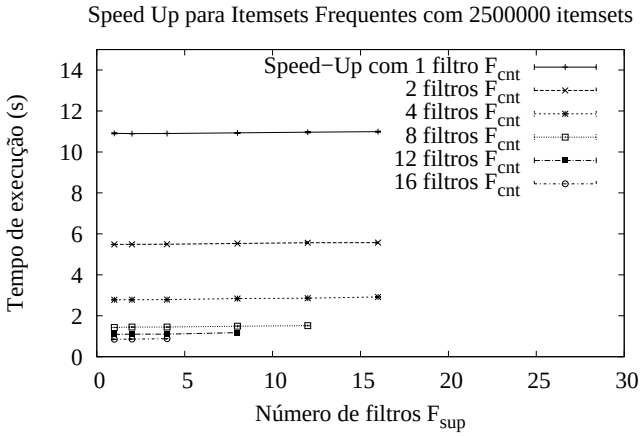
As figuras 5.1a, 5.2a e 5.3a mostram as curvas de *Tempo de execução* × *Número de filtros* F_{cnt}. São apresentadas 6 curvas em cada uma das figuras, sendo cada uma das curvas com um número diferente de filtros F_{sup}. Cada uma das figuras mostra execução do algoritmo para as 3 bases escolhidas.

O tempo de execução do algoritmo depende claramente, do número de instâncias do *filtro* F_{cnt}. Cada uma das instâncias do *filtro* F_{cnt} itera sobre os pontos e faz a contagem local de frequência. Cada uma das instâncias do *filtro* F_{sup} cuida de um candidato. Conforme pode ser visto nas figuras 5.1b, 5.2b e 5.3b, o número de instâncias do *filtros* F_{sup} não afeta o tempo de execução, pois o tempo gasto nelas é muito pequeno em comparação com as instâncias do *filtro* F_{cnt}. O tempo de execução poderá ser afetado pelo número de instâncias desse filtro, se o número de candidatos for grande o suficiente para ser comparável ao número de pontos, assim, necessitando de uma divisão da contagem.

[1] O filtro de leitura poderia ser instanciado mais de uma vez para diminuir o tempo total de leitura de dados, dividindo os dados entre vários computadores, mas como o tempo de leitura também foi subtraído do tempo total de execução, optou-se por não aumentar o número de instâncias.

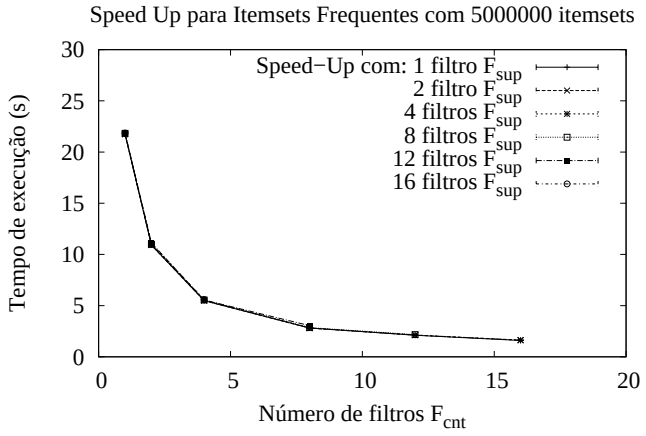


(a) *filtro* F_{cnt}

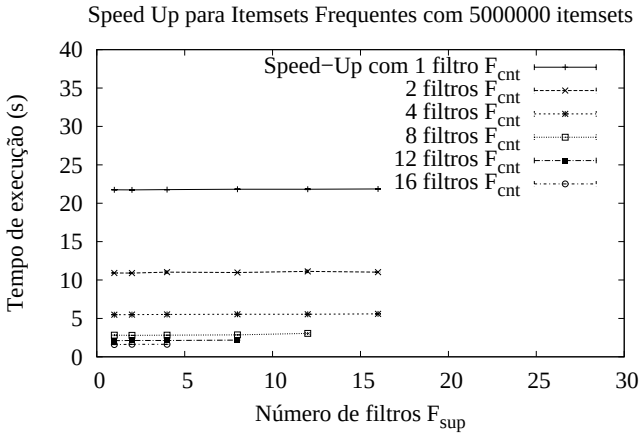


(b) *filtro* F_{cnt}

Figura 5.1: *Speed-up* do algoritmo Frequent itemsets com 2,5 milhões de pontos para *filtro* F_{cnt} e *filtro* F_{sup}



(a) *filtro* F_{cnt}



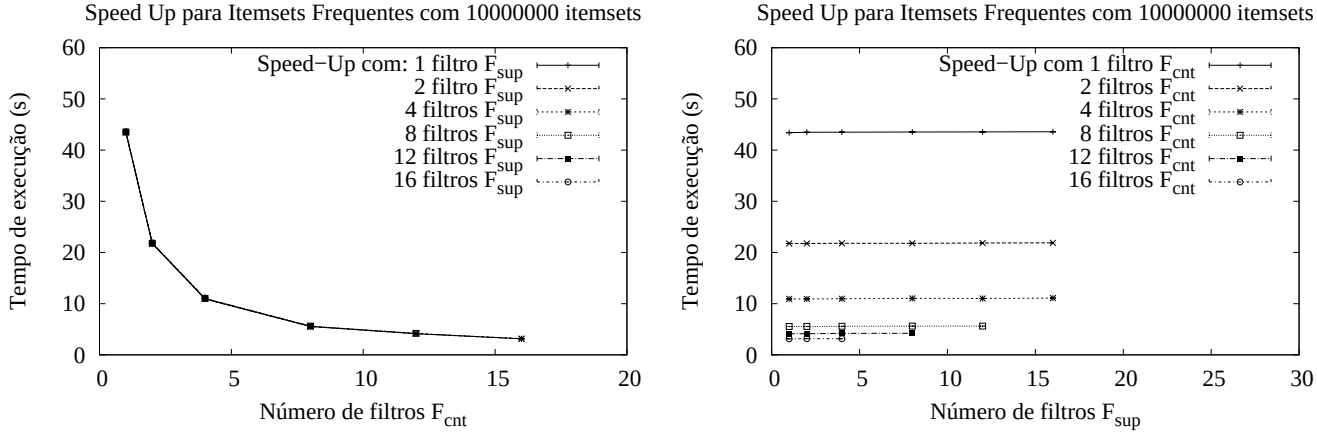
(b) *filtro* F_{cnt}

Figura 5.2: *Speed-up* do algoritmo Frequent itemsets com 5 milhões de pontos para *filtro* F_{cnt} e *filtro* F_{sup}

A tabela 5.2 mostra o *speed-up* alcançado com a variação do número de *filtros* F_{cnt}. Essa tabela foi construída utilizando o menor tempo de execução alcançado para 1 instância do *filtro* F_{cnt} e o maior tempo de execução alcançado para as outras instâncias do *filtro* F_{cnt} (ambos os casos, escolhendo entre todas as execuções do *filtro* F_{sup}). Com o aumento de número de *filtros* F_{cnt} o crescimento do *speed-up* diminui, pois o *overhead* de criação da utilização de mais instâncias de filtros começam a influenciar no tempo da computação. O aumento do número de pontos permite o aumento da quantidade de

Número de Filtros F_{cnt}	2500000 pontos		5000000 pontos		10000000 pontos	
	tempo(s)	*speed-up*	tempo(s)	*speed-up*	tempo(s)	*speed-up*
1	10,89	–	21,74	–	43,41	–
2	5,57	1,95	11,12	1,95	21,89	1,98
4	2,91	3,73	5,59	3,88	11,08	3,91
8	1,52	7,16	3,04	7,13	5,61	7,73
12	1,17	9,26	2,17	10,00	4,21	10,30
16	0,87	12,40	1,63	13,32	3,16	13,72

Tabela 5.2: *Speed-up* para *filtro* F_{cnt} do algoritmo Frequent itemsets



(a) *filtro* F_{cnt} (b) *filtro* F_{cnt}

Figura 5.3: *Speed-up* do algoritmo Frequent itemsets com 10 milhões de pontos para *filtro* F_{cnt} e *filtro* F_{sup}

computação em cada instância, diminuindo a influência desse *overhead*. A tabela 5.3 complementa esses resultados, mostrando que o número de instâncias do *filtro* F_{sup} não altera o *speed-up* do algoritmo de maneira significativa.

Número de Filtros F_{sup}	2500000 pontos		5000000 pontos		10000000 pontos	
	tempo(s)	*speed-up*	tempo(s)	*speed-up*	tempo(s)	*speed-up*
1	10,90	–	21,75	–	43,41	–
2	10,89	1,00	21,74	1,00	43,47	0,99
4	10,90	1,00	21,76	0,99	43,49	0,99
8	10,93	0,99	21,82	0,99	43,52	0,99
12	10,96	0,99	21,82	0,99	43,55	0,99
16	10,99	0,99	21,86	0,99	43,57	0,99

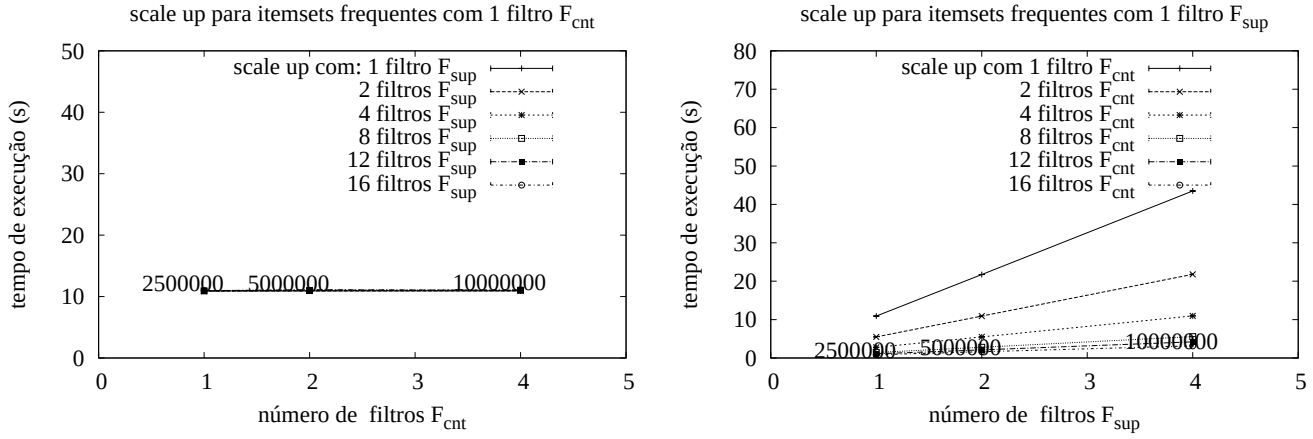
Tabela 5.3: *Speed-up* para *filtro* F_{sup} do algoritmo Frequent itemsets com 1 *filtro* F_{cnt}

Em termos de escalabilidade, os filtros se comportam dentro do esperado. O *filtro* F_{cnt} tem, no pior caso, uma eficiência de 87%. Apesar disso, o algoritmo se mostrou escalável enquanto a quantidade de pontos por processador se manteve num patamar de 100 mil pontos. A tabela 5.4 apresenta esses resultados.

Número inicial de Filtros F_{cnt}	2500000 pontos		5000000 pontos		10000000 pontos	
	tempo(s)	*scale-up*	tempo(s)	*scale-up*	tempo(s)	*scale-up*
1	10,89	–	11,12	0,97	11,08	0,98
2	5,48	–	5,56	0,98	5,61	0,97
4	2,77	–	2,82	0,98	3,16	0,87

Tabela 5.4: *Scale-up* para *filtro* F_{cnt} do algoritmo Frequent itemsets

As curvas apresentadas nas figuras 5.4a, 5.5a e 5.6a mostram a falta de influência do número de filtros F_{sup} na escalabilidade do algoritmo.

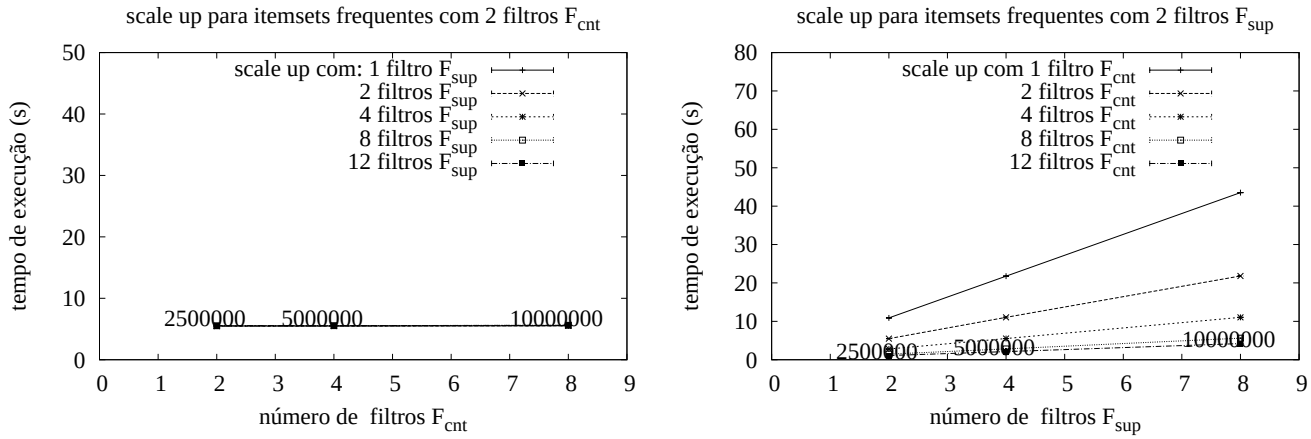


(a) 1 *filtro* F_{cnt} (b) 1 *filtro* F_{sup}

Figura 5.4: Scale Up do algoritmo Frequent itemsets iniciando com 1 *filtro* F_{cnt} *e* F_{sup}

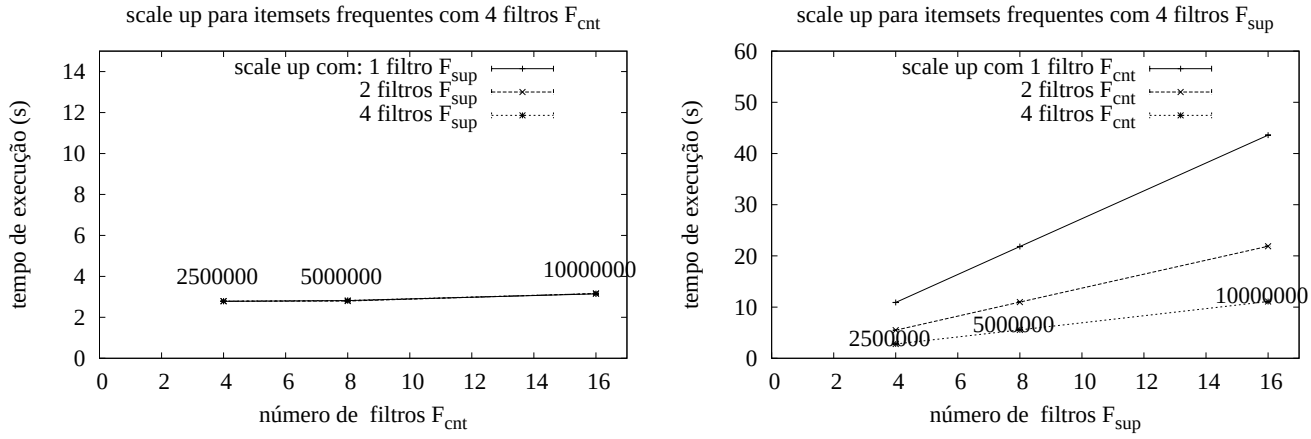
As curvas apresentadas nas figuras 5.4b, 5.5b e 5.6b mostram a deficiência do *filtro* F_{cnt} em suportar o aumento do tamanho da base de entrada, pois ele itera sobre todos os elementos da base. É possível observar que o aumento do número de *filtros* F_{sup} não é capaz de manter a eficiência do algoritmo.

O algoritmo Frequent itemsets tem sua eficiência claramente dependente do número de instâncias do *filtro* F_{cnt}.



(a) 2 *filtros* F_{cnt} (b) 2 *filtros* F_{sup}

Figura 5.5: Scale Up do algoritmo Frequent itemsets iniciando com 2 *filtros* F_{cnt} *e* F_{sup}



(a) 4 *filtros* F_{cnt} (b) 4 *filtros* F_{sup}

Figura 5.6: Scale Up do algoritmo Frequent itemsets iniciando com 4 *filtros* F_{cnt} *e* F_{sup}

5.1.2 K-means

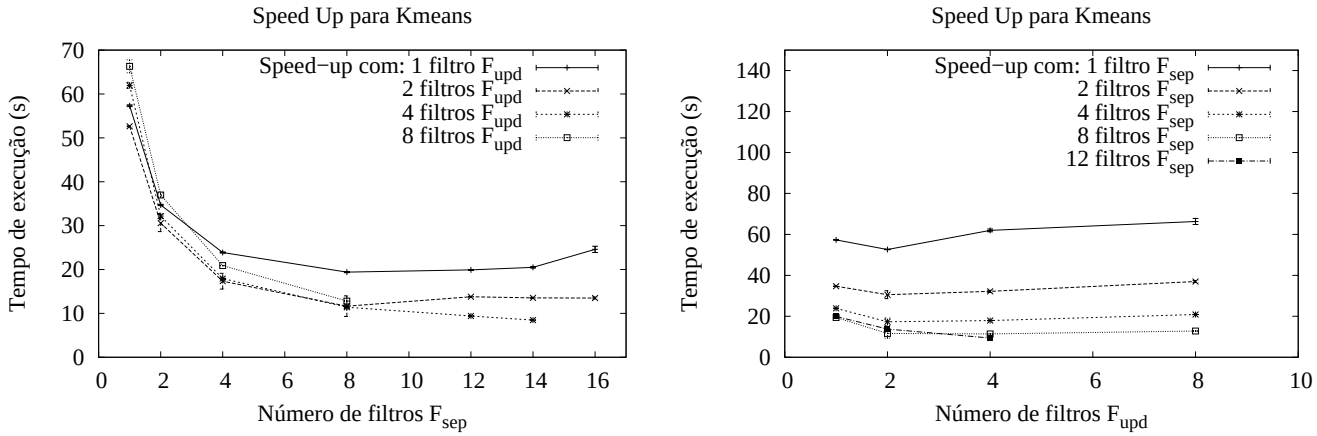
O algoritmo K-means possui os filtros de leitura e escrita de dados, e os filtros que foram apresentados na seção 4.1, e foram instanciados várias vezes, para realizar a avaliação da aplicação.

A tabela 5.5 mostra todas as variações que foram utilizadas para realizar os experimentos, cada uma das combinações possíveis foi executada. O número máximo de instâncias do *filtro* F_{upd} é determinado pelo número de centros escolhido no início do algoritmo, pois cada uma delas deve ser responsável por pelo menos um centro.

Optou-se por executar com 8 centróides.

Filtro/Recurso:	Variação:
Filtro F_{sep}	1;2;4;8;12 e 16 instâncias
Filtro F_{upd}	1;2;4;8 instâncias
Base de dados	50; 100 e 200 milhares de pontos com duas dimensões cada.

Tabela 5.5: Variações para execução do algoritmo K-means

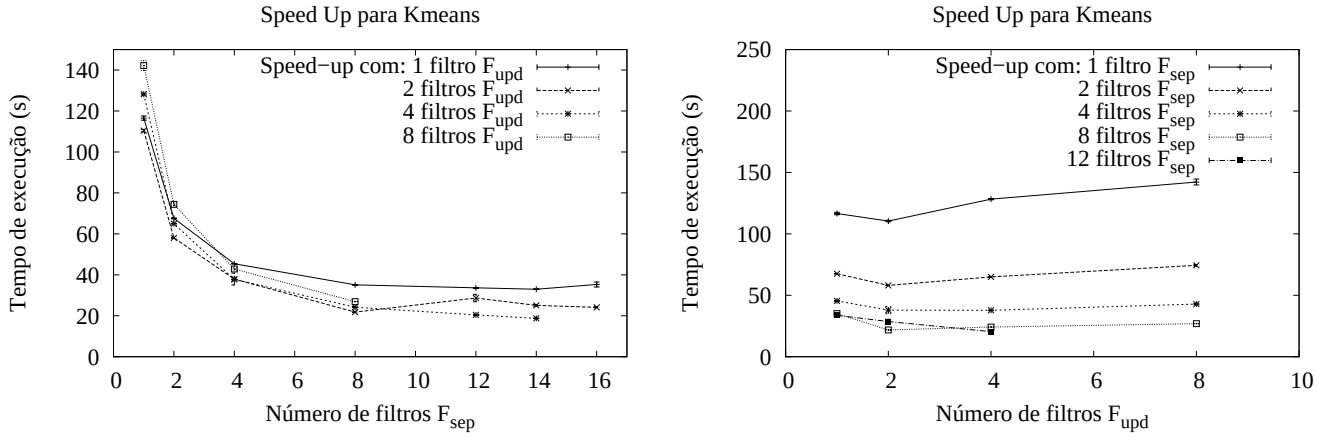


(a) *filtro* F_{sep} (b) *filtro* F_{upd}

Figura 5.7: Gráficos de *speed-up* do algoritmo K-means com 50 mil pontos para *filtro* F_{sep} e *filtro* F_{upd}

As figuras 5.7a, 5.8a e 5.9a mostram o comportamento do tempo de execução do algoritmo quando ocorre variação no número de *filtros* F_{sep}. Há queda do tempo de execução a medida que aumenta a quantidade de *filtros* F_{sep}, sendo que essa diminuição se torna menos evidente a medida que a quantidade de pontos por instância diminui (os pontos são divididos entre as instâncias do *filtro* F_{sep}). No gráfico 5.7a é possível verificar que o tempo de execução aumenta quando passa de 8 para 12 instâncias, sendo que o mesmo não acontece no gráfico 5.9a. Nas curvas que mostram o tempo de execução com 1 e 2 *filtros* F_{upd} nota-se claramente esse aumento.

É importante ressaltar nesses gráficos que as curvas que mostram o tempo de execução com 8 *filtros* F_{upd} possuem tempo maior do que aquelas com menor número de *filtros* F_{upd}, isso para execuções de 1 e 2 *filtros* F_{sep}. A partir de 4 *filtros* F_{sep} nota-se que as curvas se cruzam, mostrando que o tempo de execução depende do número de instâncias de ambos os filtros.



(a) *filtro* F_{sep} (b) *filtro* F_{upd}

Figura 5.8: Gráficos de *speed-up* do algoritmo K-means com 100 mil pontos para *filtro* F_{sep} e *filtro* F_{upd}

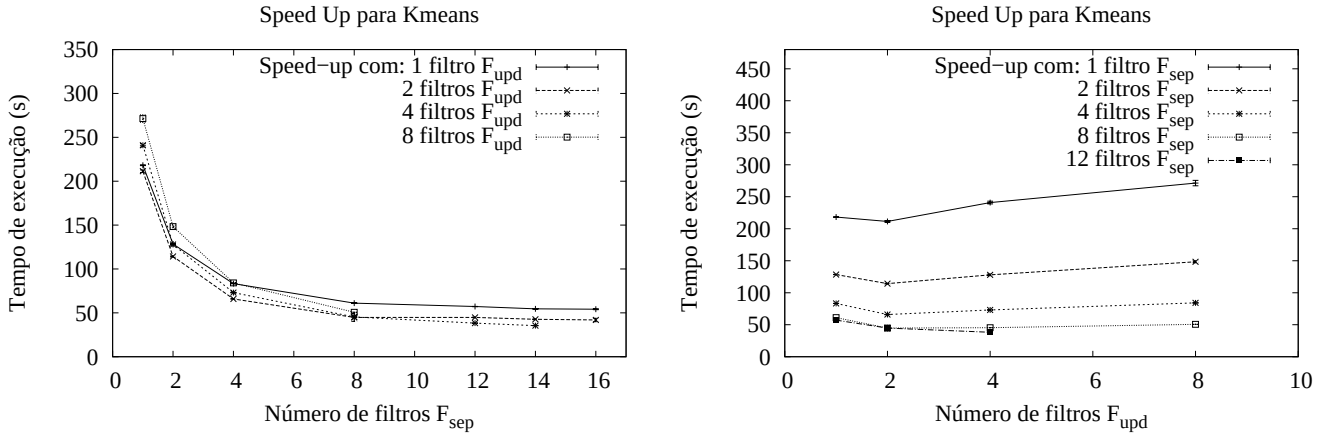
Número de Filtros F_{sep}	50000 pontos		100000 pontos		200000 pontos	
	tempo(s)	*speed-up*	tempo(s)	*speed-up*	tempo(s)	*speed-up*
1	57,32	–	116,54	–	218,23	–
2	34,69	1,65	67,57	1,72	128,42	1,69
4	23,88	2,40	45,41	2,56	83,31	2,61
8	19,40	2,95	35,10	3,31	61,08	3,57
12	19,91	2,87	33,64	3,46	57,18	3,81
16	24,58	2,33	35,28	3,30	54,16	4,02

Tabela 5.6: *Speed-up* para *filtro* F_{sep} do algoritmo K-means, com 1 *filtro* F_{upd}

Como cada instância do *filtro* F_{upd} recebe os pontos de cada um dos centros, quando o número de instâncias de *filtros* F_{sep} aumenta, o número de mensagens recebidas aumenta, necessitando um maior tempo de processamento, consequentemente gerando um gargalo.

Número de Filtros F_{sep}	50000 pontos		100000 pontos		200000 pontos	
	tempo(s)	*speed-up*	tempo(s)	*speed-up*	tempo(s)	*speed-up*
1	61,95	–	128,24	–	240,94	–
2	32,16	1,92	64,97	1,97	127,90	1,88
4	17,92	3,45	37,75	3,39	73,08	3,29
8	11,41	5,42	24,17	5,30	45,32	5,31
12	9,41	6,58	20,41	6,28	38,27	6,29

Tabela 5.7: *Speed-up* para *filtro* F_{sep} do algoritmo K-means, com 4 *filtros* F_{upd}



(a) *filtro* F_{sep} (b) *filtro* F_{upd}

Figura 5.9: Gráficos de *speed-up* do algoritmo K-means com 100 mil pontos para *filtro* F_{sep} e *filtro* F_{upd}

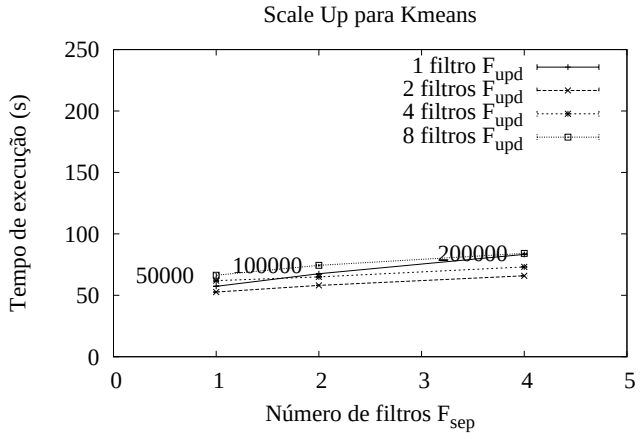
A tabela 5.6 mostra como o *speed-up* diminui seu crescimento com o aumento do número de instâncias do *filtro* F_{sep}, mantendo o número de *filtros* F_{upd} constante. Há inclusive um decaimento no *speed-up* com o aumento para 16 instâncias do *filtro* F_{sep}. O *speed-up* tem um crescimento maior quando o número de instâncias do *filtro* F_{upd} é maior, conforme pode ser visto na tabela 5.7.

As figuras 5.7b, 5.8b e 5.9b mostram o comportamento do tempo de execução do algoritmo quando ocorre variação no número de *filtros* F_{upd}. Esses gráficos mostram que somente quando o número de *filtros* F_{sep} é grande, vale a pena instanciar mais *filtros* F_{upd}, causando aumento efetivo no *speed-up*. Além disso, nota-se claramente que o aumento do número de *filtros* F_{sep} diminui o tempo de execução.

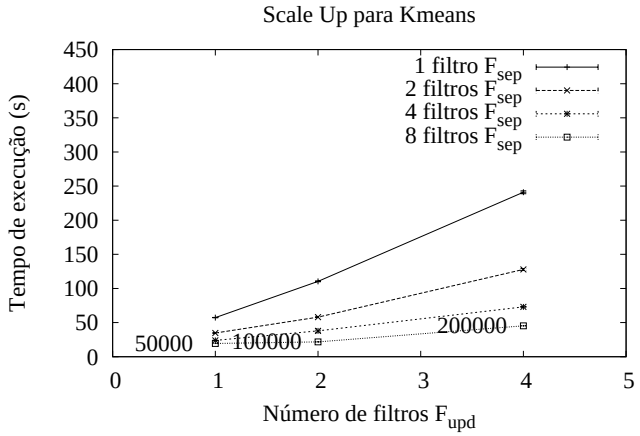
A implementação realizada do algoritmo utiliza um *bitmap* dos pontos para informar se os pontos pertencem ou não aquele centro. Essa mensagem possuí um tamanho fixo, dependente apenas do tamanho da base de dados de entrada dividido pelo número de instâncias do *filtro* F_{sep}. Dessa forma, o aumento do número de instâncias do *filtro* F_{upd} somente afeta positivamente o algoritmo se existirem muitas instâncias do *filtro* F_{sep}, pois a fragmentação da mensagem favorece o envio, nesse caso.

Em termos de escalabilidade, o algoritmo é eficiente quando o número de instâncias do *filtro* F_{sep} aumenta, conforme pode ser visto nas figuras 5.10a, 5.11a e 5.12a, mas utilizando somente 1 *filtro* F_{upd} nota-se queda na eficiência do algoritmo, que fica bem claro no gráfico 5.10a.

Assim como no algoritmo Frequent itemsets, no K-means há uma clara dependência da eficiência do algoritmo com o aumento do número de *filtros* F_{sep}. Isso fica

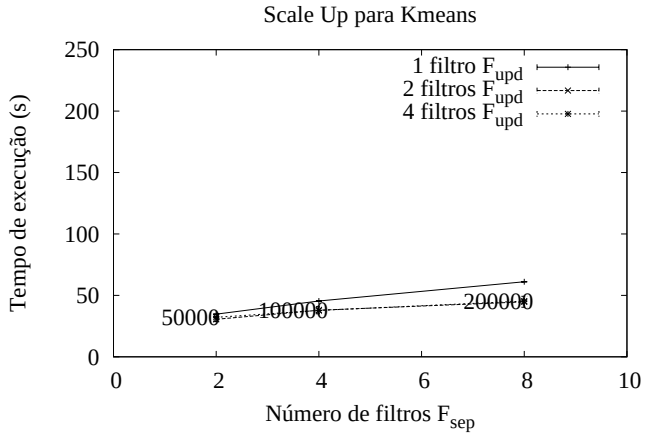


(a) 1 *filtro* F_{sep}

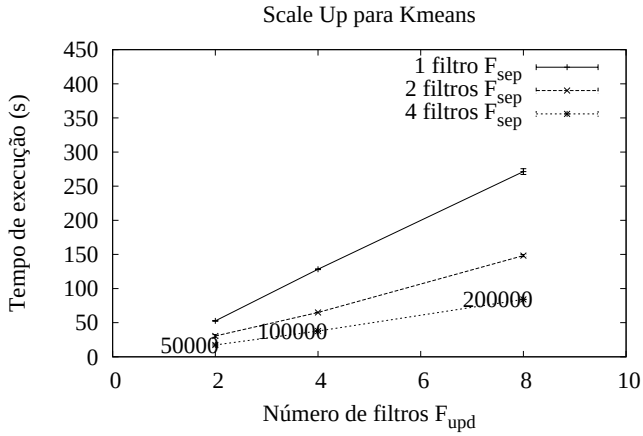


(b) 1 *filtro* F_{upd}

Figura 5.10: Gráficos de Scale Up do algoritmo K-means iniciando com 1 *filtro* F_{sep} *e* F_{upd}



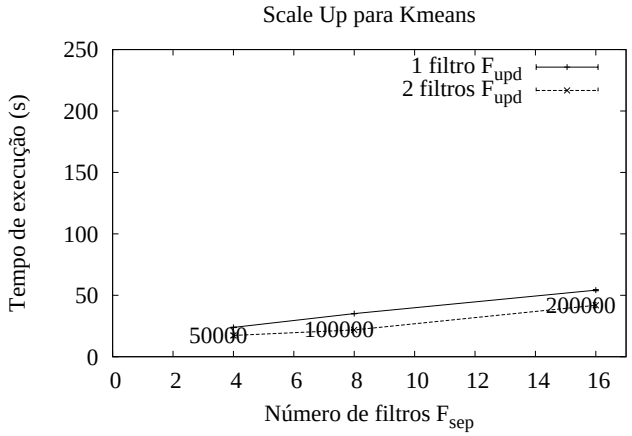
(a) 2 *filtros* F_{sep}



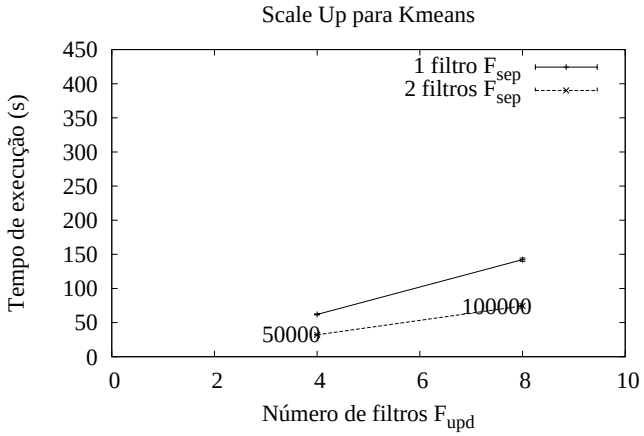
(b) 2 *filtros* F_{upd}

Figura 5.11: Gráficos de Scale Up do algoritmo K-means iniciando com 2 *filtros* F_{sep} *e* F_{upd}

bastante evidente ao visualizar os gráficos 5.10b, 5.11b e 5.12b. As curvas que mais evidenciam esse comportamento, são as curvas para 1 e 8 *filtros* F_{sep}, do gráfico 5.10b. Esse efeito também se deve ao fato da mensagem que vai do *filtro* F_{sep} para o *filtro* F_{upd} contendo os pontos de cada centro ser formada por um *bitmap* e ter tamanho definido pelo número de pontos da base de entrada dividido pelo número de instâncias do *filtro* F_{sep}. O aumento no tempo de execução causado quando o número de instâncias do *filtro* F_{upd} é grande pode ser explicado pelo fato do crescimento no número de mensagens



(a) 4 *filtros* F_{sep}



(b) 4 *filtros* F_{upd}

Figura 5.12: Gráficos de Scale Up do algoritmo K-means iniciando com 4 *filtros* F_{sep} *e* F_{upd}

a serem trafegadas no pipeline dos filtros, além das instâncias desse filtro ficarem boa parte do tempo ociosas, comparando com as instâncias do *filtro* F_{sep}.

5.1.3 K-nearest neighbors

O algoritmo K-nearest neighbors possui os filtros de leitura e escrita de dados, e o filtro raiz, que não precisam de ser executados com mais de uma instância [2]. Além desses, existem os filtros que foram apresentados na seção 4.3.

Os experimentos foram executados variando o número de instâncias do *filtro* F_{dst} e do *filtro* F_{knn}.

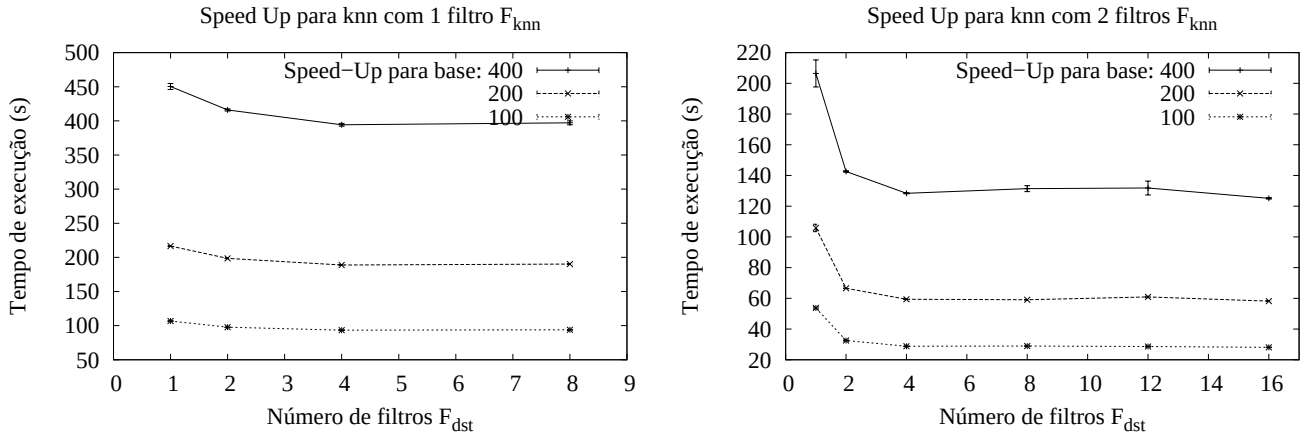
Filtro/Recurso:	Variação:
Filtro F_{dst}	1;2;4;8;12 e 16 instâncias
Filtro F_{knn}	1;2;4;8;12 e 16 instâncias
Base de dados de treino	10000 pontos
Base de dados de teste	100, 200 e 400 pontos

Tabela 5.8: Variações para execução do algoritmo K-nearest neighbors

A tabela 5.8 mostra todas as variações que foram utilizadas para realizar os experimentos, cada uma das combinações possíveis foi executada. As bases de dados foram geradas aleatoriamente, e elas possuem 10 dimensões cada e existem 20 classes

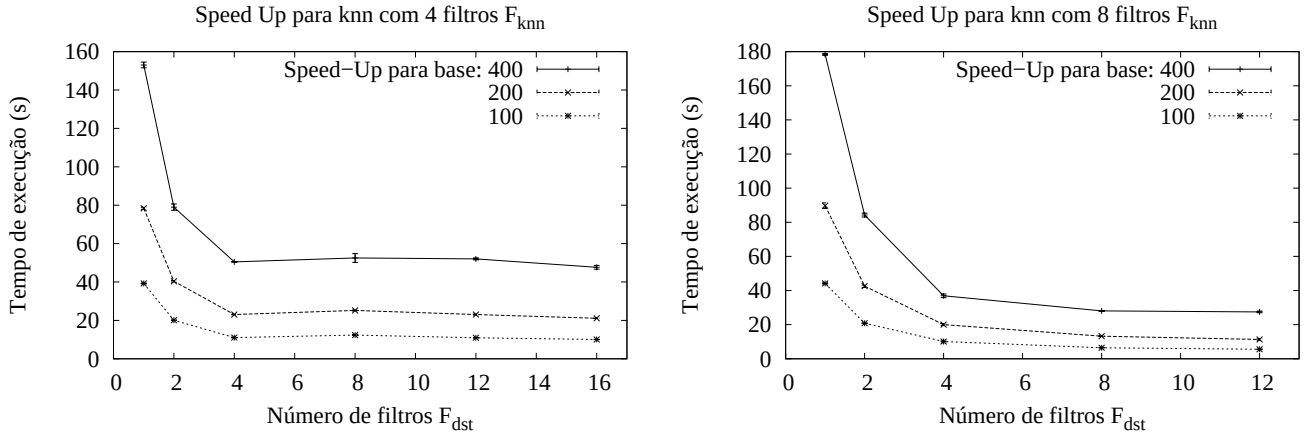
[2]Mesmo caso dos algoritmo anteriores

possíveis para cada um dos pontos pertencer. O número de vizinhos escolhido para classificar os pontos de teste foi 40.



(a) 1 *filtro* F_{knn} (b) 2 *filtros* F_{knn}

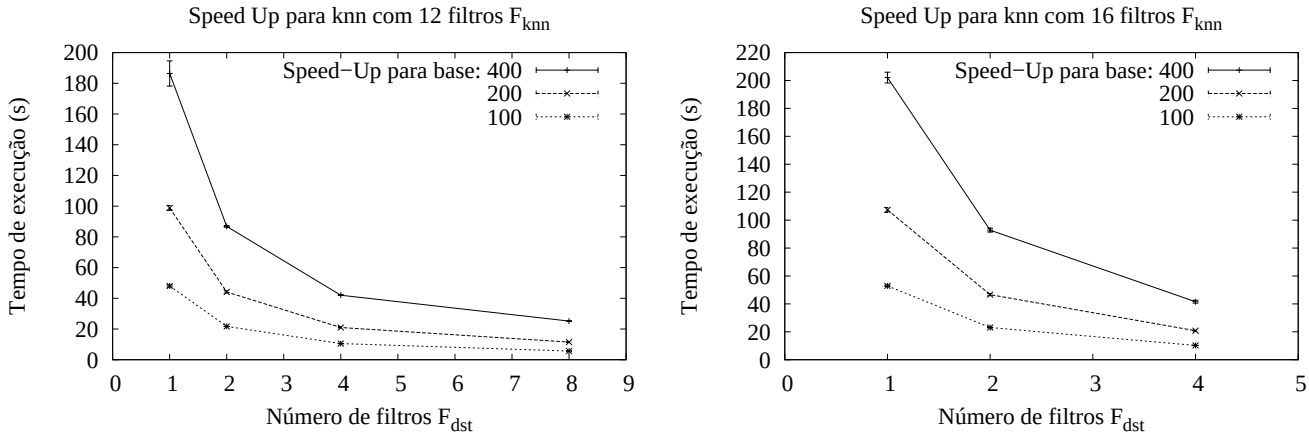
Figura 5.13: *Speed-up* do algoritmo K-nearest neighbors para *filtro* F_{dst} com 1 e 2 *filtros* F_{knn}



(a) 4 *filtro* F_{knn} (b) 8 *filtros* F_{knn}

Figura 5.14: *Speed-up* do algoritmo K-nearest neighbors para *filtro* F_{dst} com 4 e 8 *filtros* F_{knn}

O processo de ordenação, que é realizado no *filtro* F_{knn}, é a operação que mais pesa no tempo de execução do algoritmo. Nota-se essa dependência, quando se compara o tempo de execução dos gráficos das figura 5.13, 5.14 e 5.15 entre os gráficos. No gráfico 5.13a há uma pequena diminuição no tempo de execução com o aumento do



(a) 12 *filtro* F_{knn}

(b) 16 *filtros* F_{knn}

Figura 5.15: *Speed-up* do algoritmo K-nearest neighbors para *filtro* F_{dst} com 12 e 16 *filtros* F_{knn}

número de *filtros* F_{dst}, já nos outros gráficos, esse aumento causa uma variação mais significativa, mostrando o gargalo que essa operação causa no algoritmo.

Nota-se que a partir de 4 instâncias do *filtro* F_{dst} não há diminuição significativa no tempo de execução, ocorrendo até um aumento desse tempo em alguns casos.

O tempo de execução também decai com o aumento do número de *filtros* F_{dst} nos gráficos da figura 5.15, mas na figura 5.15a nota-se que o tempo de execução começa a sofrer de overhead causado por aumento no número de instâncias do *filtro* F_{knn}.

A tabela 5.9 mostra como o *speed-up* do aumento de *filtros* F_{dst} cresce mais a medida que o número de *filtros* F_{knn} aumenta.

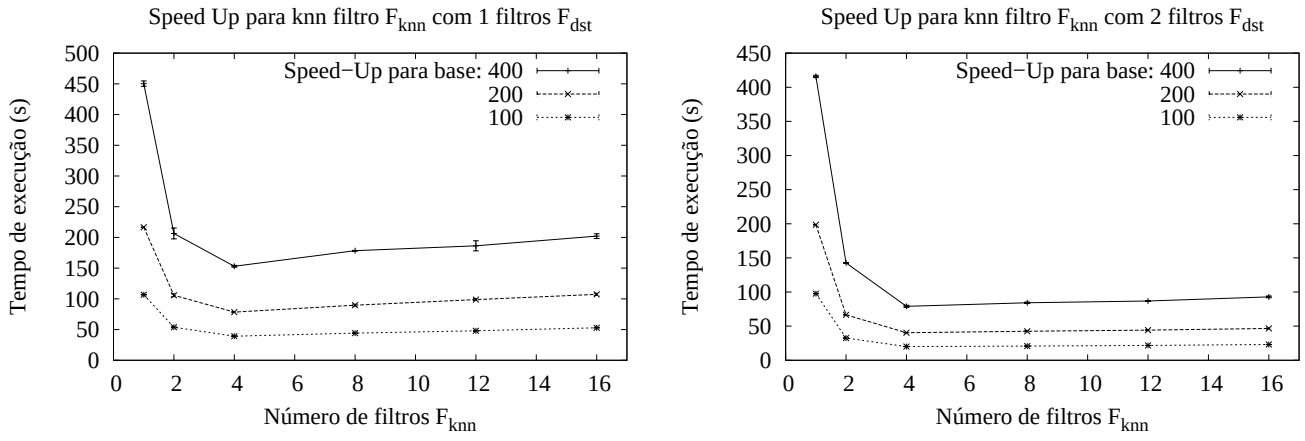
Conforme dito anteriormente, o aumento do número de *filtros* F_{knn} é mais impactante no tempo de execução do algoritmo. As figuras 5.16, 5.17 e 5.18ilustram esse comportamento. Existem, porém, um caso particular onde o número de *filtros* F_{dst} é bastante impactante no *speed-up* do algoritmo, esse caso está ilustrado no gráfico 5.17b.

A tabela 5.10 mostra esse comportamento. Para 8 instâncias do *filtro* F_{dst} o *speed-up* do algoritmo chega um valor super-linear, comparando apenas com o número de instâncias do *filtro* F_{knn}. Essa super-linearidade na verdade não existe quando considera-se todos as instâncias de filtros envolvidas, pois no caso do K-nearest neighbors, não existe um gargalo bem definido em uma determinada instância, como ocorre nos outros dois algoritmos.

Os gráficos apresentados na figura 5.19 mostram que o aumento do número de instâncias do *filtro* F_{dst} só tornam o algoritmo escalável para números elevados de instâncias de *filtros* F_{knn}.

Número de *filtros* F_{dst}	1 filtro F_{knn}		2 *filtros* F_{knn}		4 *filtros* F_{knn}	
	tempo(s)	*speed-up*	tempo(s)	*speed-up*	tempo(s)	*speed-up*
1	450,37	–	206,44	–	153,03	–
2	415,95	1,08	142,59	1,44	78,99	1,93
4	394,24	1,14	128,36	1,60	50,48	3,03
8	397,09	1,13	131,45	1,57	52,49	2,91
12	–	–	131,83	1,56	52,04	2,94
16	–	–	125,13	1,64	47,65	3,21
Número de *filtros* F_{dst}	8 *filtros* F_{knn}		12 *filtros* F_{knn}		16 *filtros* F_{knn}	
	tempo(s)	*speed-up*	tempo(s)	*speed-up*	tempo(s)	*speed-up*
1	178,36	–	186,38	–	202,08	–
2	84,23	2,11	86,80	2,14	92,83	2,17
4	36,91	4,83	42,02	4,43	41,54	4,86
8	28,04	6,36	25,13	7,41	–	–
12	27,46	6,49	–	–	–	–

Tabela 5.9: *Speed-up* para *filtro* F_{dst} do algoritmo K-nearest neighbors com 400 pontos de teste

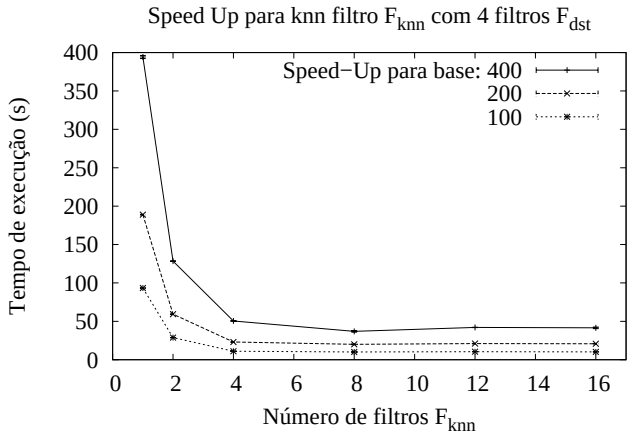


(a) 1 *filtro* F_{dst}

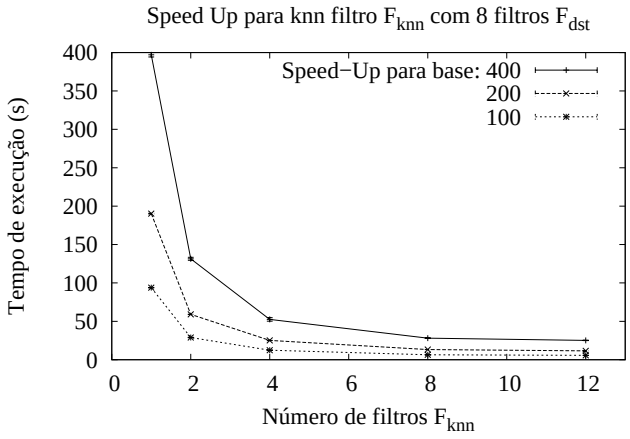
(b) 2 *filtros* F_{dst}

Figura 5.16: *Speed-up* do algoritmo K-nearest neighbors para *filtro* F_{knn} com 1 e 2 *filtros* F_{dst}

No caso dos gráficos apresentados na figura 5.20, o algoritmo possui uma escalabilidade super-linear para o aumento do número de instâncias do *filtro* F_{knn}. Essa super-linearidade corrobora com a informação dita anteriormente, que o algoritmo não possui um gargalo como os outros dois algoritmos, tornando o tempo de execução total dependente do número total de instâncias de todos os filtros.

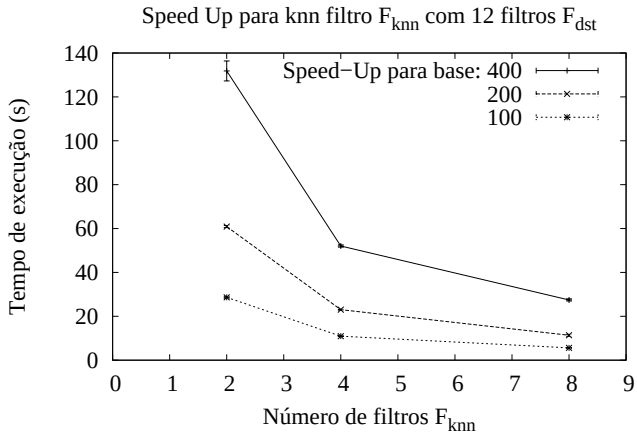


(a) 4 *filtros* F_{dst}

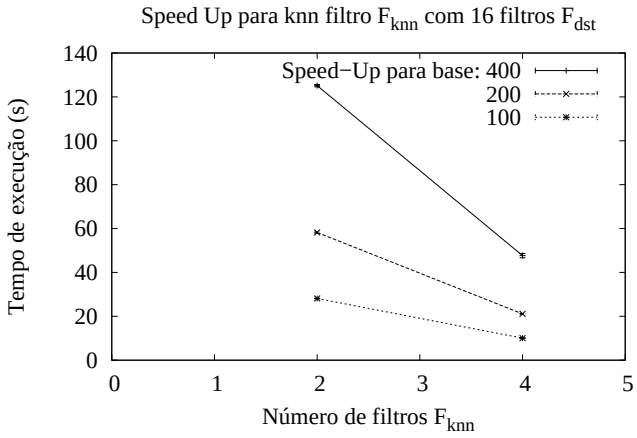


(b) 8 *filtros* F_{dst}

Figura 5.17: *Speed-up* do algoritmo K-nearest neighbors para *filtro* F_{knn} com 4 e 8 *filtros* F_{dst}



(a) 12 *filtros* F_{dst}



(b) 16 *filtros* F_{dst}

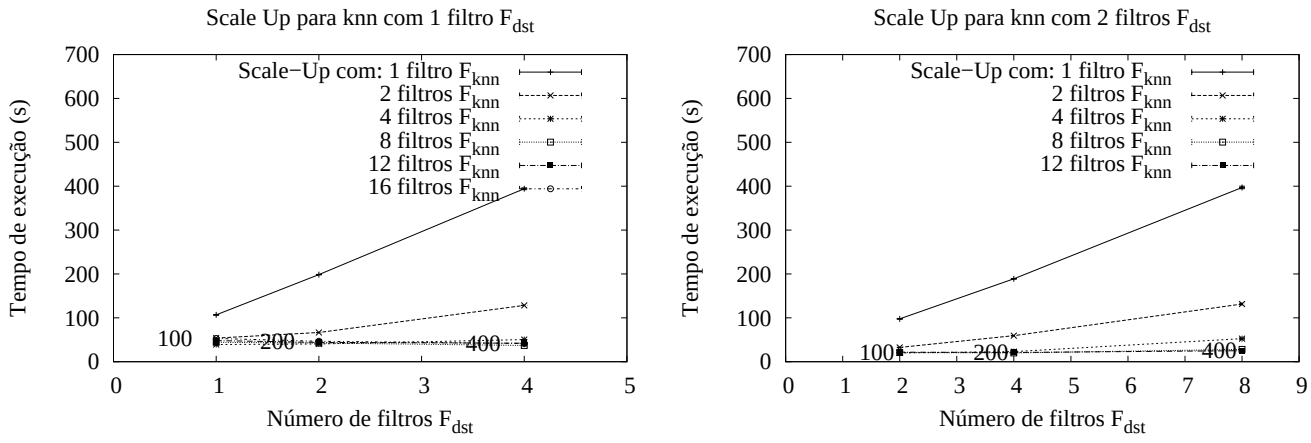
Figura 5.18: *Speed-up* do algoritmo K-nearest neighbors para *filtro* F_{knn} com 12 e 16 *filtros* F_{dst}

5.2 Comparação com a versão manual

Nessa seção são apresentados os gráficos comparando o tempo de execução dos algoritmos desenvolvidos por um programador e os algoritmos gerados automaticamente com o tradutor apresentado nessa dissertação. Os experimentos foram executados no mesmo ambiente apresentado anteriormente, sendo realizada uma nova rodada de experimentos.

Número de *filtros* F_{knn}	1 filtro F_{dst}		2 *filtros* F_{dst}		4 *filtros* F_{dst}	
	tempo(s)	*speed-up*	tempo(s)	*speed-up*	tempo(s)	*speed-up*
1	450,37	–	415,95	–	394,24	–
2	206,44	2,18	142,59	2,91	128,36	3,07
4	153,03	2,94	78,99	5,26	50,48	7,80
8	178,36	2,52	84,23	4,93	36,91	10,67
12	186,38	2,41	86,80	4,79	42,02	9,38
16	202,08	2,22	92,83	4,48	41,54	9,49
Número de *filtros* F_{knn}	8 *filtros* F_{dst}		12 *filtros* F_{dst}		16 *filtros* F_{dst}	
	tempo(s)	*speed-up*	tempo(s)	*speed-up*	tempo(s)	*speed-up*
1	397,09	–	263,67	–	250,27	–
2	131,45	3,02	131,83	2,00	125,13	2,00
4	52,49	7,56	52,04	5,06	47,65	5,25
8	28,04	14,16	27,46	9,60	–	–
12	25,13	15,79	–	–	–	–

Tabela 5.10: *Speed-up* para *filtro* F_{knn} do algoritmo K-nearest neighbors com 400 pontos de teste



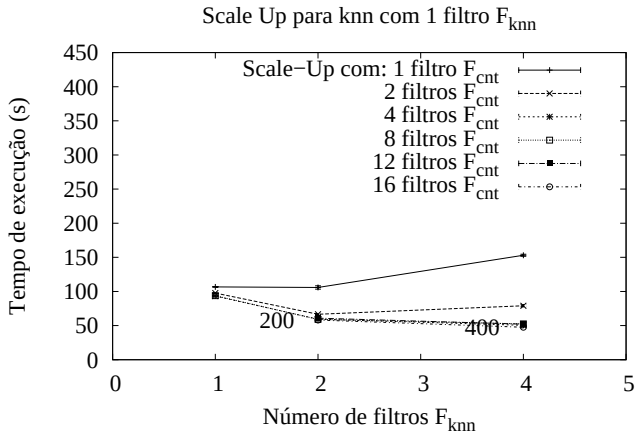
(a) 1 *filtro* F_{dst}

(b) 2 *filtros* F_{dst}

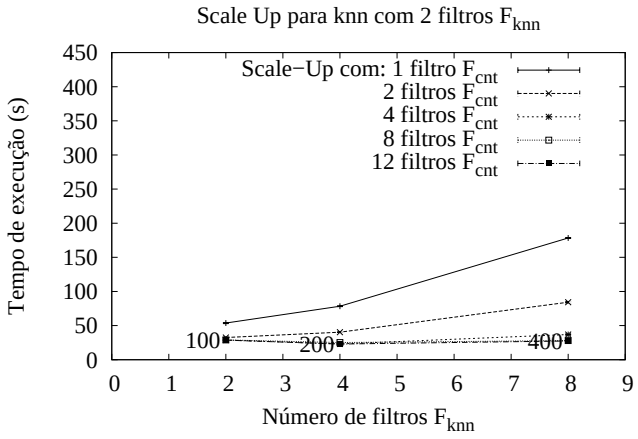
Figura 5.19: Scale Up do algoritmo K-nearest neighbors *filtro* F_{dst} iniciando com 1 e 2 *filtros* F_{dst}

5.2.1 Frequent Itemsets

A versão manual implementada para o algoritmo Frequent itemsets possui 2 filtros que realizam a computação, os filtro F_{cnt} que é responsável por contar o número de itens que possuem os candidatos a frequentes na sua partição dos dados e o filtro F_{sup} que faz a agregação do resultado local. A variação do número de instâncias desses filtros é comparada respectivamente com a variação do número de filtros F_{cnt} e F_{sup} da versão



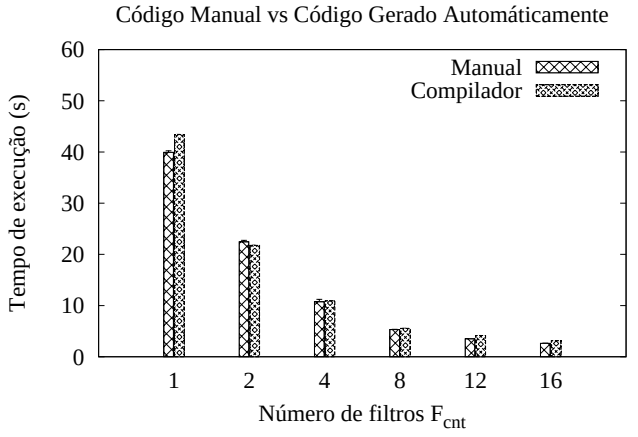
(a) 1 *filtro* F_{knn}



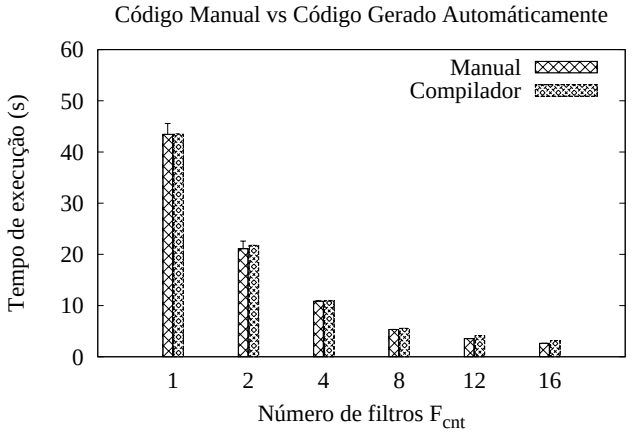
(b) 2 *filtros* F_{knn}

Figura 5.20: Scale Up do algoritmo K-nearest neighbors *filtro* F_{knn} iniciando com 1 e 2 *filtros* F_{knn}

gerada automaticamente, pois esses filtros realizam tarefas semelhantes do algoritmo.



(a) 1 *filtro* F_{sup}

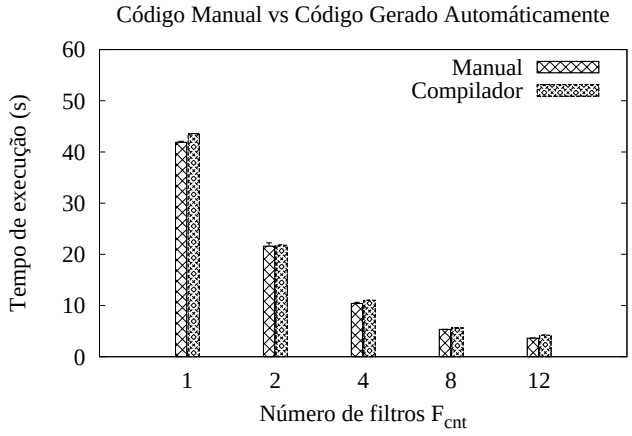


(b) 2 *filtros* F_{sup}

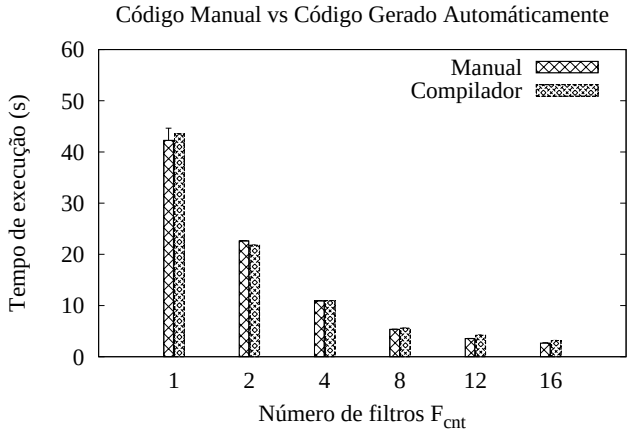
Figura 5.21: Comparação do tempo de execução entre a versão Compilada e a versão Manual do Frequent itemsets para 1 e 2 *filtros* F_{sup}

O algoritmo foi executado com a base de dados contendo *10 milhões* de pontos com 20 dimensões cada.

Os gráficos apresentados nas figuras 5.21, 5.22 e 5.23 mostram o tempo de execução dos algoritmos variando o número de filtros F_{cnt} (e filtro Adder no caso da versão Manual). Nota-se que não há diferença significativa nos tempos de execução em relação a ambas implementações. Vale ressaltar que mesmos nos casos onde a média do

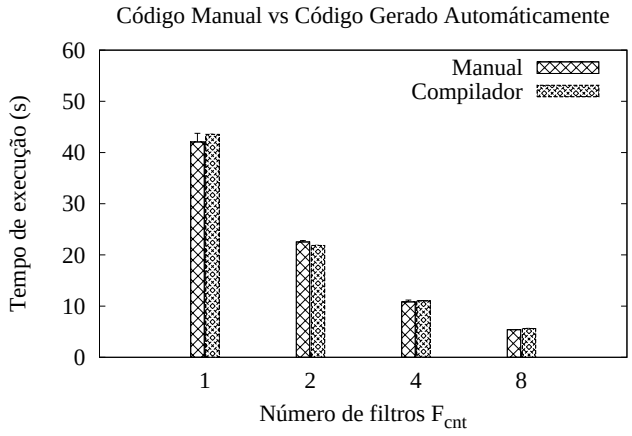


(a) 8 *filtros* F_{sup}

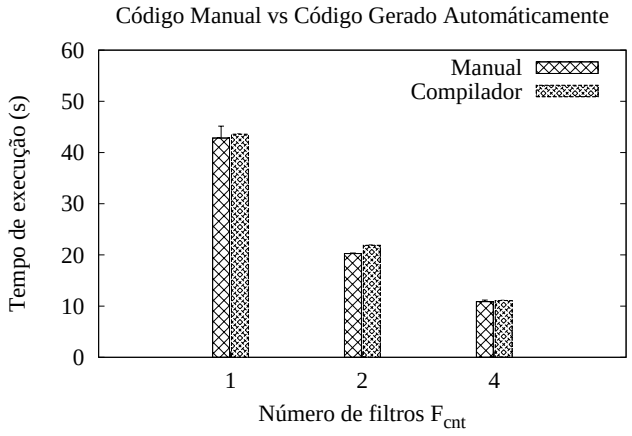


(b) 4 *filtros* F_{sup}

Figura 5.22: Comparação do tempo de execução entre a versão Compilada e a versão Manual do Frequent itemsets para 4 e 8 *filtros* F_{sup}



(a) 12 *filtros* F_{sup}

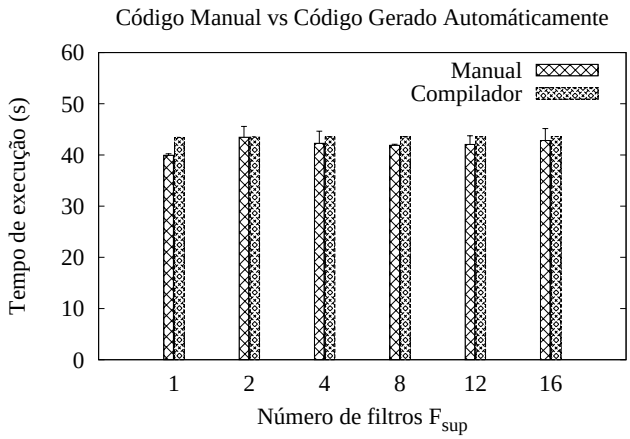


(b) 16 *filtros* F_{sup}

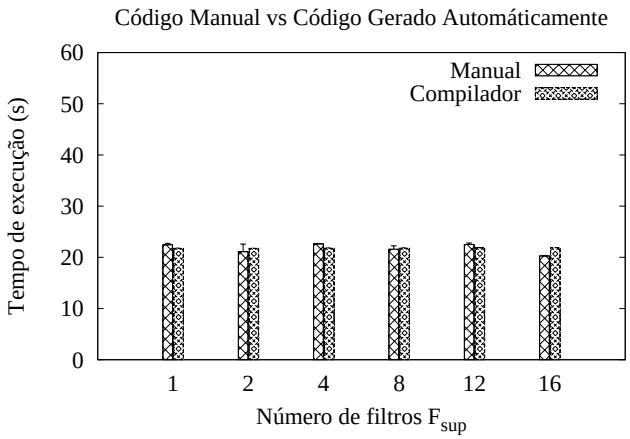
Figura 5.23: Comparação do tempo de execução entre a versão Compilada e a versão Manual do Frequent itemsets para 12 e 16 *filtros* F_{sup}

tempo de execução do algoritmo gerado automaticamente pelo tradutor foi menor que o algoritmo manual, o desvio padrão dos tempos medidos é maior que a diferença entre os tempos.

Os gráficos apresentados nas figuras 5.24, 5.25 e 5.26 mostram o tempo de execução dos algoritmos variando o número de filtros F_{sup} (e filtros Merger no caso da versão Manual). Assim como na versão gerada automaticamente pelo tradutor, o filtro Merger não influência significativamente no tempo de execução do algoritmo. Porém

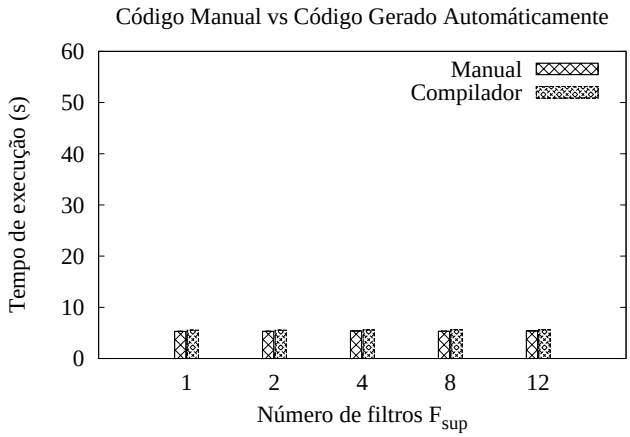


(a) 1 *filtro* F_{cnt}

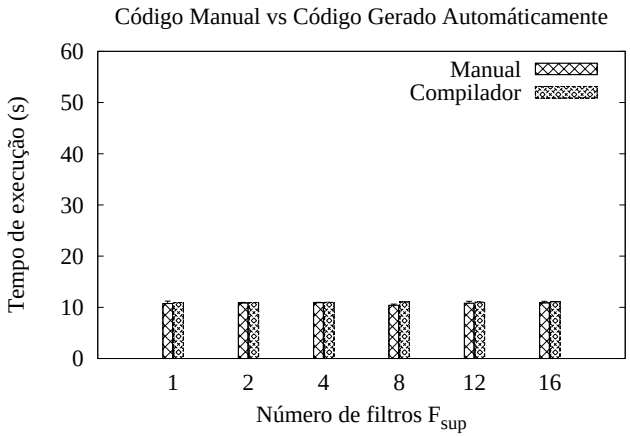


(b) 2 *filtros* F_{cnt}

Figura 5.24: Comparação do tempo de execução entre a versão Compilada e a versão Manual do Frequent itemsets com 1 e 2 *filtros* F_{cnt}



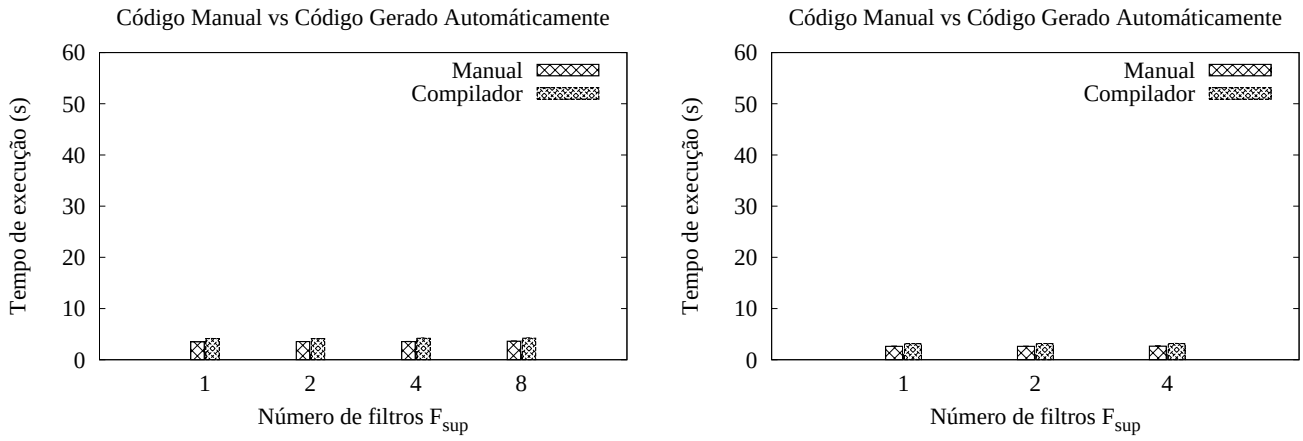
(a) 8 *filtros* F_{cnt}



(b) 4 *filtros* F_{cnt}

Figura 5.25: Comparação do tempo de execução entre a versão Compilada e a versão Manual do Frequent itemsets com 4 e 8 *filtros* F_{cnt}

nota-se que na maior parte dos casos o algoritmo gerado automaticamente tem tempo de execução maior que o algoritmo feito manualmente. Para o algoritmo Frequent itemsets a versão gerada automaticamente pelo tradutor obteve um desempenho bem parecido com o algoritmo desenvolvido manualmente.



(a) 12 *filtros* F_{cnt}

(b) 16 *filtros* F_{cnt}

Figura 5.26: Comparação do tempo de execução entre a versão Compilada e a versão Manual do Frequent itemsets com 12 e 16 *filtros* F_{cnt}

5.2.2 K-means

A versão manual implementada para o algoritmo K-means possui 2 filtros que realizam a computação, o filtro Distance que é responsável por calcular a distância dos pontos até os centros e o filtro Centroids que é responsável por calcular no novo centro. Esse filtros são comparados respectivamente aos filtros implementados gerados automaticamente pelo tradutor.

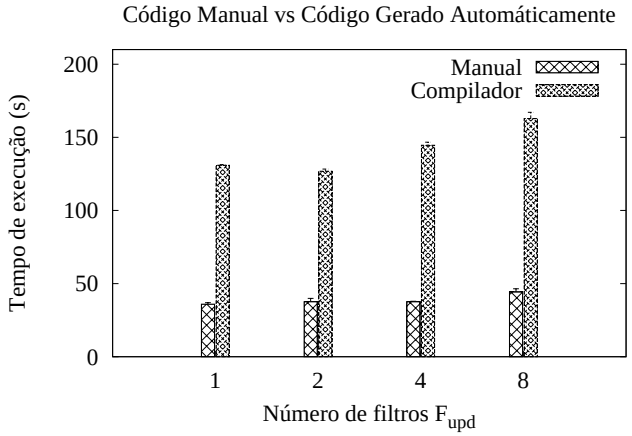
Variou-se o número de filtros para medir o tempo de execução do algoritmo para a base de dados com 200 mil pontos. O número de centros a ser calculado é de 8.

Os gráficos apresentados nas figuras 5.27 e 5.28 mostram claramente que o tempo de execução do algoritmo para a versão gerada pelo tradutor é superior ao tempo de execução da versão manual quando o número de filtros F_{upd} é pequeno. Levando-se em consideração os gráficos apresentados, a variação no número de instâncias do filtro F_{upd} piora o tempo de execução do algoritmo.

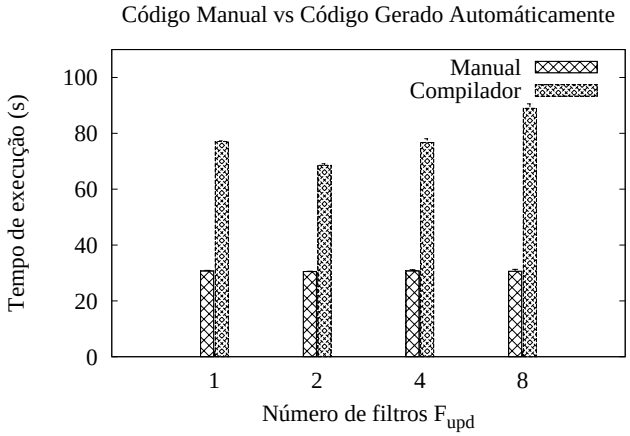
O gráfico 5.29 mostra que o algoritmo gerado automaticamente tem melhor desempenho com o aumento do número de filtros F_{upd}.

As figuras 5.30 e 5.31 deixam claro que o algoritmo gerado manualmente tem desempenho superior ao gerado automaticamente.

Comparando-se a tendências dos gráficos nota-se que o algoritmo gerado automaticamente é mais escalável que o feito manualmente. Essa escalabilidade pode ser explicada pelo fato do algoritmo feito manualmente utilizar técnicas mais sofisticadas para o calculo dos novos centros, evitando um número de execuções do loop principal

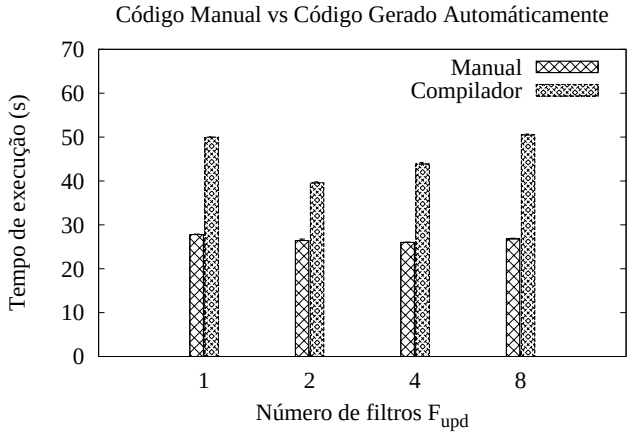


(a) 1 *filtro* F_{sep}

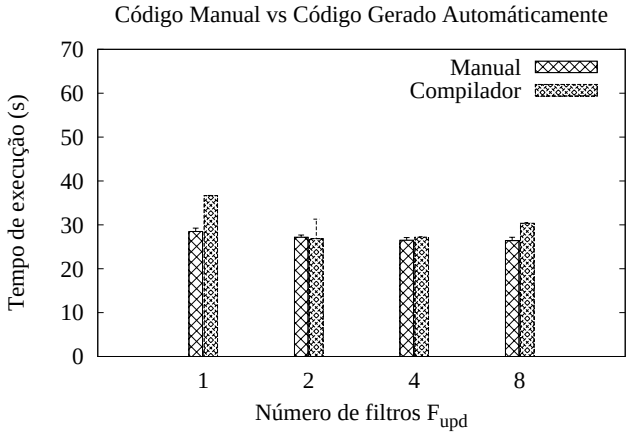


(b) 2 *filtros* F_{sep}

Figura 5.27: Comparação do tempo de execução entre a versão Compilada e a versão Manual do K-means com 1 e 2 *filtros* F_{sep}



(a) 4 *filtros* F_{sep}



(b) 8 *filtros* F_{sep}

Figura 5.28: Comparação do tempo de execução entre a versão Compilada e a versão Manual do K-means com 4 e 8 *filtros* F_{sep}

do K-means maior do que na versão gerada automaticamente.

5.2.3 K-nearest neighbors

A versão manual implementada para o algoritmo K-nearest neighbors possui apenas 1 filtro que realiza a computação. A base é copiada localmente em cada um desses filtros e cada um deles executa o calculo para um ponto de teste, um típico *bag of tasks*. Como na versão gerada automaticamente desse algoritmo a computação é realizada

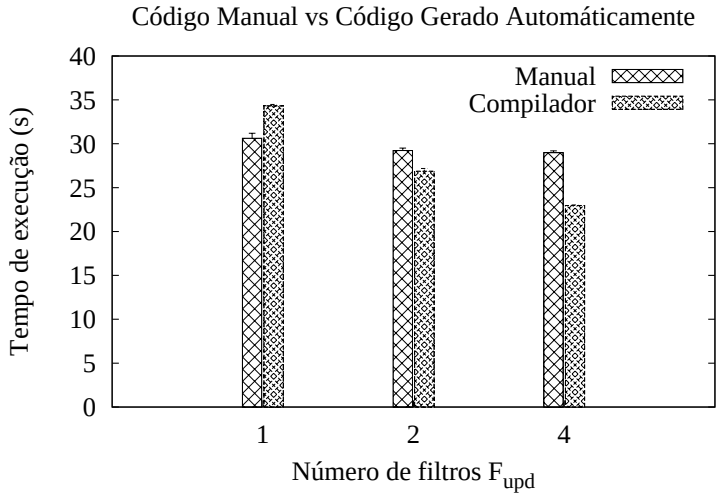
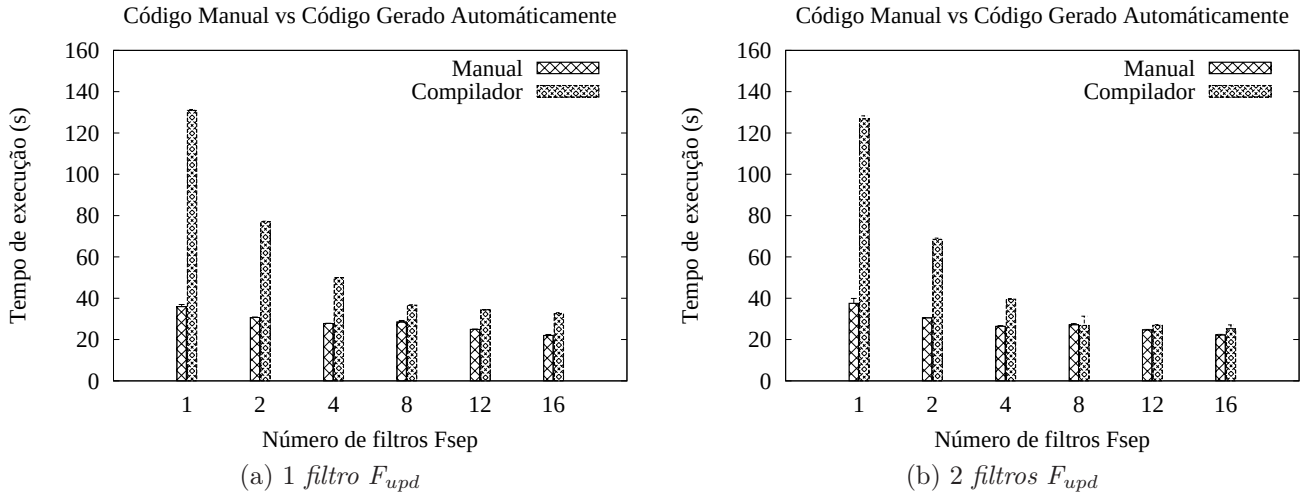


Figura 5.29: Comparação do tempo de execução entre a versão Compilada e a versão Manual do K-means com 12 *filtros* F_{sep}



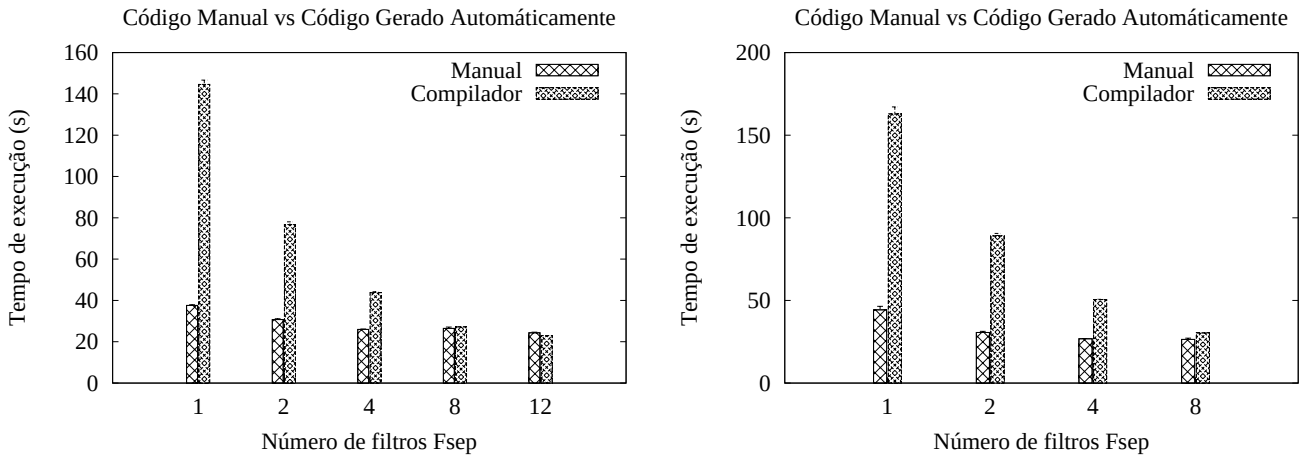
(a) 1 *filtro* F_{upd} (b) 2 *filtros* F_{upd}

Figura 5.30: Comparação do tempo de execução entre a versão Compilada e a versão Manual do K-means

em ambos os filtros (F_{dst} e F_{knn}) a melhor comparação a ser realizada entre as duas versões implementadas deve considerar o tempo de execução quando o número de filtros F_{dst} e F_{knn} é o mesmo.

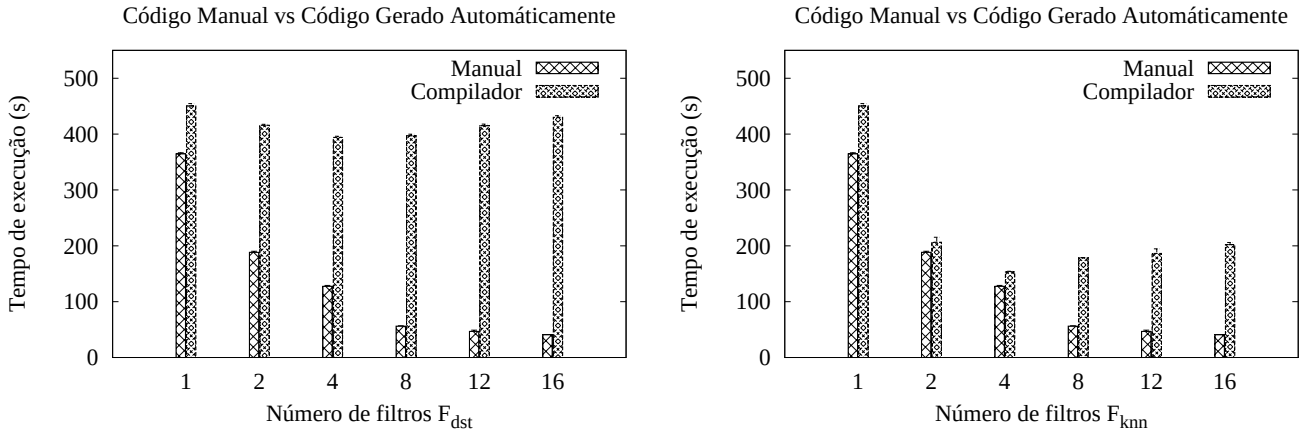
A título de exemplo, os gráficos 5.32a e 5.32b foram apresentados. Eles mostram a comparação do tempo de execução das versões manual e gerada automaticamente do algoritmo, mantendo 1 *filtro* F_{knn} e 1 *filtro* F_{dst} respectivamente.

O gráfico 5.33 mostra o a comparação do tempo de execução considerando os



(a) 4 *filtros* F_{upd} (b) 8 *filtros* F_{upd}

Figura 5.31: Comparação do tempo de execução entre a versão Compilada e a versão Manual do K-means



(a) *filtro* F_{dst} (b) *filtro* F_{knn}

Figura 5.32: Comparação do tempo de execução entre a versão Compilada e a versão Manual do K-nearest neighbors

grupos de 1*filtro* F_{knn} e 1 *filtro* F_{dst}; 2 *filtros* F_{knn} e 2 *filtros* F_{dst}; 4 *filtros* F_{knn} e 4 *filtros* F_{dst}; 8 *filtros* F_{knn} e 8 *filtros* F_{dst}. Nota-se que a versão gerada automaticamente tem desempenho superior quando o número total de filtros é 8 e 16. Como a base de dados é distribuída entre instâncias do *filtro* F_{dst}, a quantidade de dados a ser lida em cada uma dessas instâncias é pequena favorecendo o uso de cache e memória virtual nesse caso.

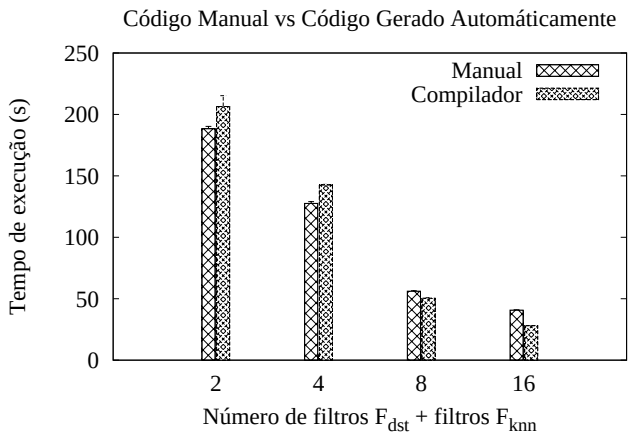


Figura 5.33: Comparação do tempo de execução entre a versão Compilada e a versão Manual do K-nearest neighbors

5.2.4 Sumário

Nesse capítulo foram apresentados os resultados dos experimentos realizados para demonstrar a qualidade dos algoritmos de paralelização automática do tradutor proposto. As três aplicações avaliadas, K-means, Frequent itemsets e K-nearest neighbors, tiveram seu desempenho medido e comparado com paralelizações implementadas manualmente, mostrando desempenho satisfatório nessa comparação. A escalabilidade dos algoritmos propostos foi avaliada variando-se o número de instâncias de filtros gerados, bem como o tamanho da entrada de dados.

Capítulo 6

Conclusões

Nessa dissertação foi apresentado um protótipo de um tradutor *fonte para fonte* que é capaz de traduzir um dialeto de *C* para execução num ambiente de computação em *Data-Flow*. Foram propostas técnicas para separação de código no *pipeline* de filtros do Anthill, bem como para a utilização das suas políticas de comunicação.

Foram avaliadas várias estratégias para geração de código para esse ambiente através da transformações de aplicações de mineração de dados. As aplicações foram paralelizadas e apresentaram melhora no tempo de execução e na sua escalabilidade.

Para o algoritmo Frequent itemsets os filtros gerados foram capazes de atingir *speed-up* de 13,72 para a execução em 16 processadores. Além de serem escaláveis, tendo no pior caso, aumento de 13% no tempo de execução.

Para os algoritmo K-means os resultados mostraram um *speed-up* de 6,58 com 12 processadores, mostrando um rendimento ruim em comparação com o algoritmo Frequent itemsets. Apesar disso, o algoritmo se mostrou escalável.

Os filtros gerados para o K-nearest neighbors que puderam ser instanciados mais de uma vez, se comportaram muito bem garantindo *speed-up* e *scale-up* do algoritmo. Diferentemente das outras duas aplicações, o código gerado para o K-nearest neighbors não possuí um gargalo importante na computação. O aumento das instâncias dos filtros avaliados contribui positivamente para diminuir o tempo de execução do algoritmo.

As estratégias para realizar a geração automática de código que foram apresentadas nesse trabalho foram capazes de gerar código escalável e com *speed-up*, dentro das limitações de cada um dos algoritmos avaliados.

Os filtros gerados pelo tradutor mostraram resultados satisfatórios de *speed-up* e *scale-up* aproveitando os recursos disponíveis para a execução da aplicação. Além disso, os resultados dos tempos de execução para os algoritmos Frequent itemsets e K-nearest neighbors gerados automaticamente foram bem próximos os algoritmos gerados

manualmente. O algoritmo K-means gerado automaticamente teve um desempenho (em termos de tempo de execução) bem inferior ao desenvolvido manualmente, mas ele suportou melhor o aumento do número de processadores disponíveis e teve até desempenho superior em alguns casos.

Comparando-se o tempo de execução dos algoritmos gerados automaticamente com versões desenvolvidas por programadores manualmente existe um ganho no uso da geração automática de filtros quando o algoritmo é favorecido pelo fato de *dividir para conquistar*, como foi o caso do algoritmo K-nearest neighbors. Quando o tradutor encontrou a mesma divisão do algoritmo utilizada pelo programador (caso do algoritmo Frequent itemsets) o comportamento de ambos foi bastante parecido.

O uso das estratégias de compilação apresentadas nesse trabalho pode permitir uma gama maior de aplicações aproveitando melhor os recursos de arquiteturas de computação que sejam distribuídas e hierárquicas e pode permitir o uso maior de ambientes de programação em Data-Flow para a solução de problemas de computação.

6.1 Trabalhos Futuros

Essa dissertação apresenta um protótipo de um tradutor que permite a entrada de um dialeto de *C*, porém grande parte das aplicações desenvolvidas não está escrita respeitando os limites utilizados para essa dissertação. Pretende-se realizar, em trabalhos futuros, o desenvolvimento de um tradutor que seja capaz de lidar com qualquer código de entrada que esteja escrito no formato de *loop* canônico apresentado no capítulo 3, mas utilizem um *C* mais completo, ou outra linguagem de programação.

Apesar do tradutor permitir a entrada de diretivas no código que descrevem as operações de redução realizadas, essas informações não foram utilizadas e a identificação automática de operações simples de redução foi implementada. A idéia é permitir ao usuário definir operações mais complexas com essas diretivas, tornando o tradutor mais poderoso.

Uma vertente interessante no processo de compilação e otimização de código, é o uso de informações de tempo de execução para facilitar o processo de transformação do código. Pretende-se utilizar esse tipo de informação para determinar o domínio de iteração de *loop*s, facilitando o uso de comunicação do tipo *labeled-stream*.

Existe um trabalho que leva em consideração o uso de um sistema de memória distribuída para armazenar os dados [Fireman et al., 2008]. Pretende-se gerar automaticamente código que seja capaz de lidar com a DSM e aproveitar os recursos que a memória distribuída pode trazer.

A ampliação do espectro de aplicações, bem como a aplicação do algoritmo em mais problemas de Data Mining é outro objetivo a ser alcançado em trabalhos futuros.

Os trabalhos [Du et al., 2003, Du & Agrawal, 2005] que descrevem o processo de geração de filtros para o modelo *filter-stream* realizando a divisão de algoritmos descritos num dialeto baseado em *Java* e que utiliza heurísticas para avaliar a quantidade de comunicação realizada e estimar qual o melhor particionamento a ser realizado. Os algoritmos propostos por esse trabalho obtiveram bons resultados na geração automática, mas os autores focam na divisão do algoritmo e não nas possibilidades diferentes de comunicação a serem utilizadas. Os autores desses trabalhos dividem o algoritmo a ser particionado nas menores unidades atômicas possíveis e as divide em filtros de forma a minimizar fatores como tempo de execução, quantidade de dados comunicados, entre outros. Esse artigo apresenta uma abordagem que pode ser utilizado nos trabalhos futuros dessa dissertação. A versão do algoritmo K-means gerada automaticamente pelo tradutor seria diretamente influenciada por essa estratégia, pois o volume de dados que trafega entre os filtros avaliados é muito grande e seria considerada a possibilidade de junção desses filtros.

Assume-se que o código a ser paralelizado está representado dentro de um *loop* no formato de *loop* canônico. A condição de parada do algoritmo é determinada pela condição do *loop while* externo. O tradutor extrai essa condição de parada em um filtro. Apesar de existir um algoritmo para detecção de condição de terminação [Ferreira et al., 2005] optou-se, nesse trabalho, por garantir a terminação através da condição declarada pelo programador, tornando o modelo um pouco mais restrito e assim evitando erros na paralelização. Um possível trabalho futuro é a modificação do tradutor para considerar essa algoritmo que detecta a condição de terminação.

Uma proposta interessante é utilizar linguagens de descrição de código paralelo, como o UPC [Consortium, 2005] ou o OpenMP [Multi-Processing, 1997], para gerar código para o Anthill, permitindo uma gama maior de aplicações a serem paralelizadas. Além disso se pode comparar códigos gerados para o Anthill com código gerados diretamente utilizando essas linguagens, pois existem compiladores para elas.

O trabalho [Teodoro et al., 2008] que apresenta o Anthill orientado a eventos é uma abordagem interessante.É possível realizar a geração de código para essa nova implementação permitindo explorar outros tipos de problemas, que são mapeados com mais clareza para o modelo de orientação a eventos.

Apêndice A

Compiladores Avaliados

Para realizar o desenvolvimento das transformações de código propostas por esse trabalho, optou-se por utilizar um compilador já existente e assim aproveitar a sua estabilidade e maturidade. A escolha desse compilador foi realizada através da avaliação de ambientes de desenvolvimento de compiladores. Buscou-se encontrar um ambiente que possuísse características que permitissem a rápida prototipagem das propostas dessa dissertação, o uso de código já existente na entrada do compilador e a possibilidade de utilização futura desse protótipo. As seguintes características foram consideradas: código fonte aberto, representação interna do código analisado em linguagem de alto-nível, transformação de seu código interno em *C* legível (mantendo *loops for, while e do*), rápida implementação de modificações no código, *curva de aprendizagem* rápida da estrutura do compilador ,além de capacidade de reutilização futura dos pacotes desenvolvidos.

Nesse capítulo são mostradas as principais características de cada um dos compiladores que foram avaliados, além dos motivos que levaram a escolha do *Cetus* [Lee et al., 2003, Johnson et al., 2004] para realizar a implementação das propostas dessa dissertação. Um resumo dessas características pode ser visto na tabela A.1. A escolha do *Cetus* como compilador implicou no uso do *antlr* como parser da linguagem, pois ambos estão integrados.

A.1 Titanium

Titanium é um dialeto de *Java* desenvolvido para aplicações de computação paralela científica que possui definições para barreiras e sincronização, mensagens ponto a ponto e *broadcast*; variáveis privatizadas; *loops* cuja ordem das iterações não interessa (*foreach*); e tipos para definição de matrizes . O compilador dessa linguagem (que

Compiladores Avaliados				
Compilador:	Escrito:	Entrada:	Saída:	Representação interna:
Titanium	*C++*	*Java*	*C* e *C++*	CFG
Scale	*Java*	*C, Fortran* e *Java*	*C* e Assembler	AST e CFG
SUIF1	*C++*	*C* e *Fortran*	formato especial SUIF1, *C* e *Assembler* Mips	Construções de baixo e alto-nível.
SUIF2	*C++*	*C, C++*, *Java* e Fortran	formato especial SUIF2, *C* e Assembler Mips	Mesmo do SUIF1
Machine SUIF	*C++*	*C, C++*, *Java* e Fortran	formato especial SUIF2, *C* e Assembler Mips	Mesmo do SUIF1 com AST e CFG
LCC	C	C	binário para várias arquiteturas	*Assembler*
Zephyr	*C++*	várias	binário para várias arquiteturas	RTL
Impact	*C++*	*C*	binário para várias arquiteturas	*Assembler*
Cosy	*C*	várias	binário para várias arquiteturas	*Assembler*
GCC	*C*	várias	binário para várias arquiteturas	RTL e AST de alto nível
LLVM	*C++*	*C, C++*	Representação interna (possui tradutor para *C*)	*Assembler*
Cil	*Ocaml*	*C*	*C*	Linguagem de alto-nível próxima ao *C*
Cetus	*Java*	*C, C++*	*C*	CFG

Tabela A.1: Resumo de características dos compiladores avaliados.

foi desenvolvido em *C++*) é capaz, através da descrição desses elementos, de gerar código *C* para a aplicação analisada. Apesar dele possuir ferramentas para análise e otimização de código bem construídas, ele necessita da implementação do código em uma linguagem de descrição de programas paralelos. Como o objetivo é realizar a tradução de um código essencialmente sequencial, optou-se por não utiliza-lo. No Titanium o código é representado como CFG, armazenando as construções básicas da linguagem [Yelick et al., 1998]

A.2 Scale

Scale ou ***S**calable **C**ompiler for **A**nalytica**l** **E**xperiments* foi construído com o intuito de ser uma ferramenta para ensino ou pesquisa em compiladores. Ele aceita como entrada códigos em *C, Fortran e Java*, sendo capaz de transformá-los para uma linguagem

intermediária comum, na qual se realiza a análise. Esse compilador gera como saída código *C* ou binário para as arquiteturas *Alpha, PowerPC ou SPARC V8*. Apesar de ser capaz de gerar *C* como esse compilador possui uma linguagem intermediária bastante simplificada, códigos gerados por ele são escritos numa versão de *C* bastante simplificada muito parecida com *Assembler*, tornando a leitura desse código complexa. Como deseja-se permitir uma fácil compreensão do código gerado na saída, optou-se por não utilizar o Scale como base para esse trabalho. O Scale armazena a AST e o CFG do código. Além disso o código é representado em SSA [for Analytical Experiments, 2006]

A.3 SUIF

SUIF foi construído para ser uma infraestrutura para pesquisa em compiladores. Ele possui a estrutura modular permitindo o uso ou o desenvolvimento de bibliotecas separadas da estrutura principal do compilador, além de permitir a entrada através de várias linguagens de programação (*C, C++, Java e Fortran*). O compilador possui uma interface que traduz o código de entrada para um formato intermediário que representa a estrutura do programa. Como o código intermediário possui diversas construções que não precisam ser avaliadas ou utilizadas nesse trabalho, optou-se por não utilizar o SUIF. Sua representação interna combina as representação de alto-nível (como *loops*, condicionais e acesso a vetores) e baixo nível (acesso a registradores, por exemplo), sendo bastante similar a uma AST. [Hall et al., 1996]

A.4 Machine SUIF

Machine SUIF é uma extensão do compilador SUIF criada para implementar otimizações específicas para determinadas arquiteturas, além de otimizações utilizando *profiling*. Apesar de possuir muitas ferramentas de análise de código embutidas, o Machine SUIF também possui uma linguagem intermediária complexa para atender os objetivos desse trabalho. A grande dificuldade de se modificá-lo está no seu objetivo de trabalhar com otimizações específicas para arquiteturas, buscando resolver problemas como alocação de registradores ou utilização de instruções de maneira mais eficiente. Utiliza a mesma representação interna do SUIF, porém armazena essas informações num CFG, organizado como SSA [Smith, 1996]

A.5 LCC

LCC é um compilador criado para ser facilmente modificado para gerar código para diferentes arquiteturas. Para atingir tal objetivo, ele possui uma linguagem intermediária de baixo-nível contendo apenas 17 funções e 36 operadores, sendo organizada como um DAG [1], bem próxima a linguagem de montagem (*Assembler*). ele é capaz de transformar código *C* padrão em código para *ALPHA, SPARC, MIPS R3000 e x86*. Mesmo permitindo a transformação para várias arquiteturas, a implementação de otimizações ou outras modificações no código mostrarem-se complexas. [Fraser & Hanson, 1991, Fraser & Hanson, 2003]

A.6 Zephyr

Zephyr é uma infraestrutura de compilação criada para com o objetivo de construir compiladores por partes. Ele não é um produto, mas um conjunto de produtos e estratégias que podem ser combinados de acordo com as necessidades do utilizador. Através dessas estratégias se pode combinar vários utilitários de compilação, como por exemplo o SUIF ou o LCC. A grande dificuldade de se implementar utilizando o Zephyr é, mais uma vez, o código intermediário, que se mostrou muito próximo a linguagem de montagem e utiliza o modelo RTL na representação.. [Appel et al., 1998]

A.7 Impact

Impact é um projeto criado para desenvolver código de qualidade para arquiteturas que permitem paralelismo em nível de instruções. Esse projeto criou o IMPACT-I, um compilador para *C* que usa técnicas avançadas de compilação para resolver problemas de execução de múltiplas instruções simultâneas. Sua representação interna possui tanto uma representação de alto-nível muito próxima ao código original em *C* e uma representação de baixo-nível em linguagem de montagem. Utiliza o CFG controle na representação interna. [Chang et al., 1991]

A.8 Cosy

Cosy é um sistema para desenvolvimento de compiladores criado pela ACE (*Associated Compiler Experts*) que foi utilizado para implementar mais de 50 compiladores comer-

[1] *Direct Acyclic Graph* ou Grafo Acíclico Direcionado

ciais, principalmente voltados para arquiteturas de sistemas embarcados. A grande dificuldade de se utilizar esse compilar é o seu tamanho, pois ele possui diversas implementações e é capaz de gerar código para várias arquiteturas diferentes. Sua representação interna é uma linguagem de montagem definida entre vários fabricantes. [compiler development system, 2006]

A.9 Gcc

GCC é um compilador criado colaborativamente para gerar código para diversas arquiteturas diferentes. Originalmente desenvolvido para *C*, hoje ele possui *front-ends* para *Java, Ada, Fortran e C++*. A grande dificuldade de implementar modificações nesse compilador está na diversidade de informações que precisam ser administradas internamente. Existem 2 representações internas no GCC, uma árvore que descreve os blocos do código, sendo quase uma representação idêntica ao código original, e uma representação de baixo-nível, o modelo RTL. [internals Manual, 2005].

A.10 LLVM

llvm ou **L**ow **L**evel **V**irtual **M**achine é um ambiente de compilação criado para permitir que implementações de novas otimizações de código possam ser realizadas sem a preocupação com a linguagem de entrada. Para alcançar esse objetivo, o projeto é dividido em 2 partes, a primeira é uma linguagem de representação intermediária baseada num processador *RISC* simples, e a segunda é uma interface de compilação que lida com essa linguagem intermediária. A grande dificuldade de se implementar modificações no llvm e o baixo nível das instruções que gera um código final muito complexo de ser analisado. [Lattner & Adve, 2004, Lattner, 2002]

A.11 Cil

Cil ou **C** **I**ntermediate **L**anguage é uma linguagem de alto-nível criada para representar código em *C* juntamente com informações do CFG. O projeto possui um compilador que transforma *C* para linguagem intermediária implementada. Ele foi criado para realizar as transformações no programa e gerar código *C* na saída. A grande dificuldade de se trabalhar com esse projeto foi a escolha da linguagem de programação do seu compilador, ele foi implementado em *Ocaml* [Remy & Vouillon, 1998] que é uma linguagem de programação Funcional da família *ML*. [Necula et al., 2002].

A.12 Cetus

Cetus é um compilador para *C, C++ e Java* criado para gerar código *C* em sua saída. Ele possui uma representação intermediária que é capaz de gerar código fiel aquele apresentado na entrada (para programas em *C*). Ele utiliza o *Antlr* como parser do código de entrada e possui uma interface pronta para geração do código intermediário. Modificações simples podem ser implementadas facilmente, desde que o programador esteja ciente de estar lidando com uma linguagem intermediária tão complexa quanto o *C*. [Lee et al., 2003, Johnson et al., 2004].

A.13 Antlr

Antlr é um parser para linguagens de programação que aceita gramáticas *LL(k)* e é capaz de reconhecer várias linguagens (*Java, C#, C++, C, Python*). O uso desse *parser* se faz necessário para realizar a análise do código de entrada. O seu uso foi recomendado pelos desenvolvedores do *Cetus*, por já existir uma integração entre essa ferramenta e ele. [Parr & Quong, 1995]

Apêndice B

Entrada e Saída do tradutor

Nesse apêndice são mostradas as Entradas e Saídas do tradutor apresentado nessa dissertação.

B.1 Frequent Itemsets

B.1.1 Entrada

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define MAX_ITEMSETS 40000
#define MAX_ITEMS 40
#define MAXLINESIZE 1024
#define MAXCOMB 1000
#define MAXCAND 1000
int main(int argc, char ** argv)
{
   int support;
   char dbFileName[100];
   int itemsets[MAX_ITEMSETS][MAX_ITEMS];
   int combinations[MAXCOMB];
   int new_candidates[MAXCAND][MAX_ITEMS];
   int n_new_cand;
   int comb_freq[MAXCOMB];
   int candidates[MAXCAND][MAX_ITEMS];
   int cand_freq[MAXCAND];
   int new_comb[MAX_ITEMS];
```

```
int nu_itemsets;
int nu_dimentions;
int temp_dimentions;
int j;
int i;
int k;
int l;
int size;
FILE * Arq;
char line[MAXLINESIZE];
char * token;
int nu_candidates;
int nu_comb;
int cardinality;
int new_cardinality;
int aux;
int aux1;
int aux2;

int m;
int n;
int cnm;
aux=0;
aux1=0;
aux2=0;

cardinality=0;
new_cardinality=0;

nu_comb=0;
nu_candidates=0;

cand_freq[0]=0;
size=0;
support=0;
dbFileName[0]=0;
itemsets[0][0]=0;
new_comb[0]=0;
combinations[0]=0;
new_candidates[0][0]=0;
n_new_cand=0;
comb_freq[0]=0;

nu_itemsets=0;
```

```
    nu_dimentions=0;
    temp_dimentions=0;

    i=0;
    j=0;
    k=0;
    l=0;


    Arq=NULL;
    line[0]=0;
    token=NULL;

    if(argc<3){
        printf("Usage: %s <database_file> <support>\n",argv[0]);
        exit(0);
    }
    strcpy(dbFileName,argv[1]);
    support=atoi(argv[2]);
#ifdef DEBUG
    printf("Arguments:\n\tdb: %s\n\tsupport: %d\n",dbFileName,support);
#endif
    {
        if ( (Arq = fopen(dbFileName,"r+") ) ){
            nu_itemsets =0;
            nu_dimentions =0;
            temp_dimentions = 0;
            for(i=0;(!feof(Arq));i=0){
                line[0]='\0';
                fgets(line,MAXLINESIZE,Arq);
                l = strlen(line);
                //remove \n char
                if(line[l-1]=='\n')l--;
                line[l]='\0';

#ifdef DEBUG
                printf("Line read (size %d): %s\n",l,line);
#endif

                if(l>0){
                    token=strtok(line," ");
                    for(i=0;(token!=NULL);i=0){
                        itemsets[nu_itemsets][temp_dimentions]=atoi(token);
```

```
#ifdef DEBUG
                    printf("token read %s\n",token);
#endif
                    temp_dimentions=temp_dimentions+1;
                    token=strtok(NULL," ");

                }
                nu_dimentions=temp_dimentions;
#ifdef DEBUG
                printf("Nu Dimentions %d\n",nu_dimentions);
#endif

                nu_itemsets=nu_itemsets+1;

                temp_dimentions=0;
                size=0;
            }

        }

#ifdef DEBUG
        printf("nu_points %d\n",nu_itemsets);
#endif


    }else{
        printf("Cannot open dbFile: %s\n",dbFileName);
        exit(0);
    }

    if(pow(2,nu_dimentions)>MAXCOMB){
        printf("MUST increase MAXCOMB to run\n");
        exit(0);
    }
    m=nu_dimentions/2;
    n=nu_dimentions;
    cnm=1;
    if (m*2 >n){
       m = n-m;
    }
    for (i=1 ; i <= m; n--, i++){

        cnm = cnm * n / i;
    }
```

```
        if(cnm>MAXCAND){
            printf("MUST increase MAXCAND to run\n");
            exit(0);
        }

        if(nu_itemsets>MAX_ITEMSETS){
            printf("MUST increase MAX_ITEMSETS to run\n");
            exit(0);
        }



        fclose(Arq);
        for (i=0;i<MAXCOMB;i++){
            for(j=0;j<MAX_ITEMS;j++){
                new_candidates[i][j]=0;
                candidates[i][j]=0;
            }
            cand_freq[i]=0;
            combinations[i]=0;
            comb_freq[i]=0;
        }
        for(j=0;j<MAX_ITEMS;j++){
            new_comb[j]=0;
        }
        nu_candidates=0;
        for(i=0;i<nu_dimentions;i++){
            candidates[i][i]=1;
            nu_candidates++;
        }
        for(i=0;i<nu_candidates;i++){
            for(j=0;j<nu_itemsets;j++){
                if (candidates[i][i]==itemsets[j][i]){
                    cand_freq[i]++;
                }
            }
        }
        for(i=0;i<nu_candidates;i++){
#ifdef DEBUG
            printf("candidate %d frequency %d\n",i,cand_freq[i]);
#endif
            aux=0;
```

```
            for(k=0;k<nu_dimentions;k++){
                if(candidates[i][k]){
                    aux=aux<<1;
                    aux+=1;
                }
                else{
                    aux=aux<<1;
                }

            }
            combinations[aux]=1;
            comb_freq[aux]=cand_freq[i];

        }
        for(i=0;i<nu_candidates-1;i++){
            if(cand_freq[i]<=support){
                break;
            }
            for(j=i+1;j<nu_candidates;j++){
                if(j==nu_candidates){
                    break;
                }
                if(cand_freq[j]<=support){
                    break;
                }
#ifdef DEBUG
                printf("NEW_CANDIDATE\n");
#endif
                for(k=0;k<nu_dimentions;k++){
                    new_comb[k]=(candidates[i][k]||candidates[j][k]);
#ifdef DEBUG
                    printf(" %d",new_comb[k]);
#endif

                }
#ifdef DEBUG
                printf("\n");
#endif

                for(k=0;k<nu_dimentions;k++){
                    new_candidates[n_new_cand][k]=new_comb[k];
                }
                n_new_cand++;
                for(k=0;k<nu_dimentions;k++){
```

```
                new_comb[k]=0;
            }
            new_cardinality=0;

        }

    }

    nu_candidates=0;
    for(i=0;i<n_new_cand;i++){
        for(k=0;k<nu_dimentions;k++){
            candidates[i][k]=new_candidates[i][k];

        }
        nu_candidates++;
    }
    for(j=0;j<nu_candidates;j++){
        cand_freq[j]=0;
    }

    cardinality=2;
}

while(nu_candidates>0){
    for(i=0;i<nu_itemsets;i++){
        for(j=0;j<nu_candidates;j++){
            for(k=0;k<nu_dimentions;k++){

                if(candidates[j][k]){
                    if(itemsets[i][k]==0){
                        break;
                    }
                }
            }
            if(k==nu_dimentions){
                cand_freq[j]++;
            }
        }
    }
    {
    new_cardinality=0;
    n_new_cand=0;
    }
    for(i=0;i<nu_candidates;i++){
```

```
#ifdef DEBUG
          printf("candidate %d frequency %d\n",i,cand_freq[i]);
#endif
          aux1=0;
          for(k=0;k<nu_dimentions;k++){
              if(candidates[i][k]){
                  aux1=aux1<<1;
                  aux1+=1;
              }
              else{
                  aux1=aux1<<1;
              }

          }
          combinations[aux1]=1;
          comb_freq[aux1]=cand_freq[i];

       }

       for(i=0;i<nu_candidates-1;i++){
          if(cand_freq[i]<=support){
              break;
          }
          for(j=i+1;j<nu_candidates;j++){
              if(j==nu_candidates){
                  break;
              }
              if(cand_freq[j]<=support)
              {
                  break;
              }
#ifdef DEBUG
              printf("NEW_CANDIDATE\n");
#endif
              aux2=0;
              for(k=0;k<nu_dimentions;k++){
                  new_comb[k]=(candidates[i][k]||candidates[j][k]);
#ifdef DEBUG
                  printf(" %d",new_comb[k]);
#endif

                  new_cardinality+=new_comb[k];
                  if(new_comb[k]){
                      aux2=aux2<<1;
```

```
                    aux2+=1;
                }
                else{
                    aux2=aux2<<1;
                }



            }
#ifdef DEBUG
            printf("\n");
#endif

            if(new_cardinality==cardinality+1&&combinations[aux2]==0){
                for(k=0;k<nu_dimentions;k++){
                    new_candidates[n_new_cand][k]=new_comb[k];
                }
                n_new_cand++;
                combinations[aux2]=1;

            }
            for(k=0;k<nu_dimentions;k++){
                new_comb[k]=0;
            }
            new_cardinality=0;

        }

    }
    for(i =0;i<nu_candidates;i++){
        cand_freq[i]=0;
    }

    nu_candidates = 0;
    for(i=0;i<n_new_cand;i++){
        for(k=0;k<nu_dimentions;k++){

            candidates[i][k]=new_candidates[i][k];
        }
        nu_candidates++;
    }


    cardinality++;
```

```
    }
    {
        nu_candidates=pow(2,nu_dimentions);
        for(i =0;i<nu_candidates;i++){
            if (combinations[i]){
                if(comb_freq[i]>support){
                    printf("comb:%d freq:%d\n",i,comb_freq[i]);
                }
            }
        }
    }


    return 0;
}
```

B.1.2 Saída

B.1.2.1 filter0.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter0.h"
int __new_iteration__;
OutputPortHandler outputstream0P;
OutputPortHandler outputstream1P;
OutputPortHandler outputstream2P;
OutputPortHandler outputstream3P;
OutputPortHandler outputstream4P;
OutputPortHandler outputstream48P;
InputPortHandler inputstream55P;
InputPortHandler inputstream5P;
int initFilter(void * work, int size)
{
```

```
    printf("Inicializing filter0\n");
    outputstream48P=dsGetOutputPortByName("outputstream48P");
    inputstream55P=dsGetInputPortByName("inputstream55P");
    inputstream5P=dsGetInputPortByName("inputstream5P");
    outputstream4P=dsGetOutputPortByName("outputstream4P");
    outputstream3P=dsGetOutputPortByName("outputstream3P");
    outputstream2P=dsGetOutputPortByName("outputstream2P");
    outputstream1P=dsGetOutputPortByName("outputstream1P");
    outputstream0P=dsGetOutputPortByName("outputstream0P");
    return 0;
}

int processFilter(void * work, int size)
{
    int nu_candidates=0;
    struct timeval tbegin,tend;
    char *instanceTime;
    instanceTime = (char *)calloc(MAXLINESIZE,sizeof(char));

    printf("Processing filter0\n");

    dsReadBuffer(inputstream5P, (& nu_candidates), sizeof(nu_candidates))
        ;
    gettimeofday(&tbegin,NULL);
    while (nu_candidates>0)
    {
        dsWriteBuffer(outputstream3P, (& __new_iteration__), sizeof(
            __new_iteration__));
        dsWriteBuffer(outputstream1P, (& __new_iteration__), sizeof(
            __new_iteration__));
        dsWriteBuffer(outputstream2P, (& __new_iteration__), sizeof(
            __new_iteration__));
        dsWriteBuffer(outputstream4P, (& __new_iteration__), sizeof(
            __new_iteration__));
        dsWriteBuffer(outputstream0P, (& __new_iteration__), sizeof(
            __new_iteration__));
        dsReadBuffer(inputstream55P, (& nu_candidates), sizeof(
            nu_candidates));
    }

    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));
```

```
    dsWriteBuffer(outputstream48P, (instanceTime), sizeof(char)*
        MAXLINESIZE);
    return 0;
}

int finalizeFilter()
{
    printf("stoping filter0\n");
    return 0;
}
```

B.1.2.2 filter0.h

```
#ifndef FILTER0_H_
#define FILTER0_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER0_H_*/
```

B.1.2.3 filter1.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter1.h"
int __new_iteration__;
InputPortHandler inputstream3P;
InputPortHandler inputstream11P;
```

```
InputPortHandler inputstream12P;
InputPortHandler inputstream18P;
InputPortHandler inputstream21P;
InputPortHandler inputstream26P;
InputPortHandler inputstream38P;
InputPortHandler inputstream42P;
OutputPortHandler outputstream49P;
OutputPortHandler outputstream47P;
OutputPortHandler outputstream56P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter1\n");
   outputstream49P=dsGetOutputPortByName("outputstream49P");
   outputstream47P=dsGetOutputPortByName("outputstream47P");
   outputstream56P=dsGetOutputPortByName("outputstream56P");
   inputstream42P=dsGetInputPortByName("inputstream42P");
   inputstream38P=dsGetInputPortByName("inputstream38P");
   inputstream26P=dsGetInputPortByName("inputstream26P");
   inputstream21P=dsGetInputPortByName("inputstream21P");
   inputstream18P=dsGetInputPortByName("inputstream18P");
   inputstream12P=dsGetInputPortByName("inputstream12P");
   inputstream11P=dsGetInputPortByName("inputstream11P");
   inputstream3P=dsGetInputPortByName("inputstream3P");
   return 0;
}

int processFilter(void * work, int size)
{

   int i;
   int ** itemsets;
   int ** candidates;
   int * cand_freq;
   struct timeval tbegin,tend;
   char *instanceTime;
   int nu_itemsets;
   int nu_dimentions;
   int j;
   int k;
   int nu_candidates;
   message_int * _new_message_int_;
   message_int_array * _new_message_int_array_;
   int  __index_candidates__ = 0 ;
   int * candidates_index ;
```

```
    int   __index_itemsets__ = 0 ;
    int * itemsets_index ;
    itemsets_index  =(int*)malloc(sizeof(int)*MAX_ITEMSETS);
    candidates_index  =(int*)malloc(sizeof(int)*MAXCAND);
    itemsets = (int **)malloc(sizeof(int*)*MAX_ITEMSETS);
    for( i = 0;i<MAX_ITEMSETS;i++){
       itemsets[i] = (int *)malloc(sizeof(int)*MAX_ITEMS);
    }

    candidates = (int **)malloc(sizeof(int*)*MAXCAND);
    for( i = 0;i<MAXCAND;i++){
       candidates[i] = (int *)malloc(sizeof(int)*MAX_ITEMS);
    }

    cand_freq = (int *)malloc(sizeof(int)*MAXCAND);


    printf("Processing filter1\n");

instanceTime = (char *)calloc(MAXLINESIZE,sizeof(char));


    _new_message_int_ = (message_int*)malloc(sizeof(message_int));

    _new_message_int_array_ = (message_int_array*)malloc(sizeof(
        message_int_array));
//using variable itemsets#5
    while ( dsReadBuffer(inputstream11P, (    _new_message_int_array_),
        sizeof( message_int_array))!= EOW ) {
       memcpy(itemsets[  __index_itemsets__ ] ,  _new_message_int_array_->
           value ,sizeof(int)*MAX_ITEMS);
       itemsets_index [  __index_itemsets__ ] =  _new_message_int_array_->
           index ;

       __index_itemsets__ ++ ;
    }


    //using variable candidates#5
    while ( dsReadBuffer(inputstream12P, (    _new_message_int_array_),
        sizeof( message_int_array))!= EOW ) {
       memcpy(candidates [  __index_candidates__ ] ,
           _new_message_int_array_->value ,sizeof(int)*MAX_ITEMS);
```

```
    candidates_index [  __index_candidates__ ] =
        _new_message_int_array_->index ;

    __index_candidates__ ++ ;
}


dsReadBuffer(inputstream21P, (& nu_dimentions), sizeof(nu_dimentions)
    );
dsReadBuffer(inputstream18P, (& nu_itemsets), sizeof(nu_itemsets));
dsReadBuffer(inputstream26P, (& nu_candidates), sizeof(nu_candidates)
    );
gettimeofday(&tbegin,NULL);
while ((dsReadBuffer(inputstream3P, (& __new_iteration__), sizeof(
    __new_iteration__))!=-2))
{
    for (i=0; i<nu_candidates; i ++ )
    {
        cand_freq[i]=0;
    }

    for (i=0; i<__index_itemsets__; i ++ )
    {
        for (j=0; j<nu_candidates; j ++ )
        {
            for (k=0; (k<nu_dimentions)&&((itemsets[i][k]!=0)||( !
                candidates[j][k])); k ++ );
            if (k==nu_dimentions)
            {
                cand_freq[j] ++ ;


            }

        }

    }

    for (j=0; j<nu_candidates; j ++ )
    {
        _new_message_int_->index=j;
        _new_message_int_->value=cand_freq[j];
```

```
            dsWriteBuffer(outputstream47P, (_new_message_int_), sizeof(
                message_int));
            dsWriteBuffer(outputstream56P, (_new_message_int_), sizeof(
                message_int));
        }
        //CORRECT recebe novos candidatos via broadcast
        dsReadBuffer(inputstream38P, (&nu_candidates), sizeof(
            nu_candidates));
        for(i=0;i<nu_candidates;i++){
            dsReadBuffer(inputstream42P, (    _new_message_int_array_),
                sizeof(message_int_array));
            memcpy(candidates[i], _new_message_int_array_->value,
                sizeof(int)*MAX_ITEMS);
            candidates_index[i] = _new_message_int_array_->index;
        }

    }

    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));

    dsWriteBuffer(outputstream49P, (instanceTime), sizeof(char)*
        MAXLINESIZE);
    return 0;
}

int finalizeFilter()
{
    printf("stoping filter1\n");
    return 0;
}
```

B.1.2.4 filter1.h

```
#ifndef FILTER1_H_
#define FILTER1_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();
```

```
#endif /*FILTER1_H_*/
```

B.1.2.5 filter2.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter2.h"
int __new_iteration__;
InputPortHandler inputstream1P;
OutputPortHandler outputstream32P;
OutputPortHandler outputstream36P;
OutputPortHandler outputstream50P;
int initFilter(void * work, int size)
{
    printf("Inicializing filter2\n");
    outputstream50P=dsGetOutputPortByName("outputstream50P");
    outputstream36P=dsGetOutputPortByName("outputstream36P");
    outputstream32P=dsGetOutputPortByName("outputstream32P");
    inputstream1P=dsGetInputPortByName("inputstream1P");
    return 0;
}

int processFilter(void * work, int size)
{
    int n_new_cand;
    int new_cardinality;
    printf("Processing filter2\n");
    struct timeval tbegin,tend;
    char *instanceTime;
    instanceTime = (char *)calloc(MAXLINESIZE,sizeof(char));


    gettimeofday(&tbegin,NULL);
```

```
    while ((dsReadBuffer(inputstream1P, (&__new_iteration__), sizeof(
        __new_iteration__))!=-2))
    {
        {
            new_cardinality=0;
            dsWriteBuffer(outputstream36P, (&new_cardinality), sizeof(
                new_cardinality));
            n_new_cand=0;
            dsWriteBuffer(outputstream32P, (&n_new_cand), sizeof(
                n_new_cand));
        }

    }

    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));

    dsWriteBuffer(outputstream50P, (instanceTime), sizeof(char)*
        MAXLINESIZE);
    return 0;
}

int finalizeFilter()
{
    printf("stoping filter2\n");
    return 0;
}
```

B.1.2.6 filter2.h

```
#ifndef FILTER2_H_
#define FILTER2_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER2_H_*/
```

B.1.2.7 filter3.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter3.h"
int __new_iteration__;
InputPortHandler inputstream2P;
InputPortHandler inputstream6P;
InputPortHandler inputstream13P;
InputPortHandler inputstream19P;
InputPortHandler inputstream22P;
InputPortHandler inputstream27P;
OutputPortHandler outputstream31P;
OutputPortHandler outputstream35P;
InputPortHandler inputstream39P;
InputPortHandler inputstream43P;
InputPortHandler inputstream56P;
OutputPortHandler outputstream51P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter3\n");
   outputstream51P=dsGetOutputPortByName("outputstream51P");
   inputstream56P=dsGetInputPortByName("inputstream56P");
   inputstream43P=dsGetInputPortByName("inputstream43P");
   inputstream39P=dsGetInputPortByName("inputstream39P");
   outputstream35P=dsGetOutputPortByName("outputstream35P");
   outputstream31P=dsGetOutputPortByName("outputstream31P");
   inputstream27P=dsGetInputPortByName("inputstream27P");
   inputstream22P=dsGetInputPortByName("inputstream22P");
   inputstream19P=dsGetInputPortByName("inputstream19P");
   inputstream13P=dsGetInputPortByName("inputstream13P");
   inputstream6P=dsGetInputPortByName("inputstream6P");
   inputstream2P=dsGetInputPortByName("inputstream2P");
   return 0;
}
```

```
int processFilter(void * work, int size)
{

   int i;
   int * combinations;
   combinations = (int *)malloc(sizeof(int)*MAXCOMB);

   int * comb_freq;
   comb_freq = (int *)malloc(sizeof(int)*MAXCOMB);

   int ** candidates;
   candidates = (int **)malloc(sizeof(int*)*MAXCAND);
   for( i = 0;i<MAXCAND;i++){
      candidates[i] = (int *)malloc(sizeof(int)*MAX_ITEMS);
   }

   int * cand_freq;
   cand_freq = (int *)malloc(sizeof(int)*MAXCAND);

   int nu_dimentions;
   int k;
   int nu_candidates;
   int total_instances;
   int my_rank;
   int my_nu_candidates;
   int aux1;
   message_int_array * _new_message_int_array_;
   _new_message_int_array_ = (message_int_array*)malloc(sizeof(
      message_int_array));

   message_int * _new_message_int_;
   _new_message_int_ = (message_int*)malloc(sizeof(message_int));

   printf("Processing filter3\n");
   struct timeval tbegin,tend;
   char *instanceTime;
   instanceTime = (char *)calloc(MAXLINESIZE,sizeof(char));

   //using variable combinations#3
   int   __index_combinations__ = 0 ;
   int * combinations_index ;
   combinations_index  =(int*)malloc(sizeof(int)*MAXCOMB);
   //using variable comb_freq#3
```

```
    int   __index_comb_freq__ = 0 ;
    int * comb_freq_index ;
    comb_freq_index  =(int*)malloc(sizeof(int)*MAXCOMB);
    while ( dsReadBuffer(inputstream6P, (    _new_message_int_), sizeof(
        message_int))!= EOW ) {
        combinations [  __index_combinations__ ] =  _new_message_int_->
            value ;
        combinations_index [  __index_combinations__ ] =  _new_message_int_
            ->index ;

        __index_combinations__ ++ ;
        if (dsReadBuffer(inputstream19P, (    _new_message_int_), sizeof(
            message_int))!= EOW ) {
        comb_freq [  __index_comb_freq__ ] =  _new_message_int_->value ;
        comb_freq_index [  __index_comb_freq__ ] =  _new_message_int_->
            index ;

        __index_comb_freq__ ++ ;
        }

    }

/*
    while ( dsReadBuffer(inputstream19P, (    _new_message_int_), sizeof(
        message_int))!= EOW ) {
        comb_freq [  __index_comb_freq__ ] =  _new_message_int_->value ;
        comb_freq_index [  __index_comb_freq__ ] =  _new_message_int_->
            index ;

        __index_comb_freq__ ++ ;
    }
*/
    //using variable candidates#5
    int   __index_candidates__ = 0 ;
    int * candidates_index ;
    candidates_index  =(int*)malloc(sizeof(int)*MAXCAND);
    while ( dsReadBuffer(inputstream13P, (    _new_message_int_array_),
        sizeof( message_int_array))!= EOW ) {
        memcpy(candidates [  __index_candidates__ ] ,
            _new_message_int_array_->value ,sizeof(int)*MAX_ITEMS);

        candidates_index [  __index_candidates__ ] =
            _new_message_int_array_->index ;
```

```
    __index_candidates__ ++ ;
}

dsReadBuffer(inputstream27P, (& nu_candidates), sizeof(nu_candidates)
    );
total_instances=dsGetTotalInstances();
my_rank=dsGetMyRank();
dsReadBuffer(inputstream22P, (& nu_dimentions), sizeof(nu_dimentions)
    );

    my_nu_candidates=(nu_candidates/total_instances);
    my_nu_candidates+= (my_rank <(nu_candidates%total_instances )) ? 1
        : 0;
gettimeofday(&tbegin,NULL);
while ((dsReadBuffer(inputstream2P, (& __new_iteration__), sizeof(
    __new_iteration__))!=-2))
{




    for(i=0;i<MAXCAND;i++){
        cand_freq[i]=0;
    }
    for(i=0;i<my_nu_candidates*dsGetNumWriters(inputstream56P);i++){
        dsReadBuffer(inputstream56P, (  _new_message_int_), sizeof(
            message_int));
        cand_freq [  _new_message_int_->index ] +=  _new_message_int_->
            value ;
    }

    for (i=0; i<my_nu_candidates; i ++ )
    {
        aux1=0;
        for (k=0; k<nu_dimentions; k ++ )
        {
            if (candidates[i][k])
            {
                aux1=(aux1<<1);
                aux1+=1;
            }
            else
            {
                aux1=(aux1<<1);
```

```
            }

        }

        combinations[aux1]=1;
        _new_message_int_->index = aux1 ;
        _new_message_int_->value = combinations[aux1] ;
        dsWriteBuffer(outputstream35P, (    _new_message_int_), sizeof(
            message_int)); ;


        comb_freq[aux1]=cand_freq[i];
        _new_message_int_->index = aux1 ;
        _new_message_int_->value = comb_freq[aux1] ;
        dsWriteBuffer(outputstream31P, (    _new_message_int_), sizeof(
            message_int)); ;
    }
    dsReadBuffer(inputstream39P, ( & nu_candidates), sizeof(
        nu_candidates));
    my_nu_candidates=(nu_candidates/total_instances);
    my_nu_candidates+= (my_rank < (nu_candidates%total_instances) ) ? 1
         : 0;

    for(i=0;i<my_nu_candidates;i++){
        dsReadBuffer(inputstream43P, (    _new_message_int_array_),
            sizeof( message_int_array));
        memcpy(candidates [  i ] , _new_message_int_array_->value ,
            sizeof(int)*MAX_ITEMS);
        candidates_index [  i ] =  _new_message_int_array_->index ;
    }

}

gettimeofday(&tend,NULL);
strcpy(instanceTime,elapsed_time(tbegin,tend));

dsWriteBuffer(outputstream51P, ( instanceTime), sizeof(char )*
    MAXLINESIZE);
return 0;
}

int finalizeFilter(  )
{
    printf("stoping filter3\n");
```

```
    return 0;
}
```

B.1.2.8 filter3.h

```
#ifndef FILTER3_H_
#define FILTER3_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER3_H_*/
```

B.1.2.9 filter4.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter4.h"
int __new_iteration__;
InputPortHandler inputstream4P;
InputPortHandler inputstream9P;
InputPortHandler inputstream14P;
InputPortHandler inputstream23P;
InputPortHandler inputstream28P;
OutputPortHandler outputstream30P;
InputPortHandler inputstream32P;
OutputPortHandler outputstream33P;
InputPortHandler inputstream35P;
InputPortHandler inputstream36P;
```

```
InputPortHandler inputstream37P;
InputPortHandler inputstream40P;
OutputPortHandler outputstream41P;
InputPortHandler inputstream44P;
InputPortHandler inputstream47P;
OutputPortHandler outputstream52P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter4\n");
   outputstream52P=dsGetOutputPortByName("outputstream52P");
   inputstream47P=dsGetInputPortByName("inputstream47P");
   inputstream44P=dsGetInputPortByName("inputstream44P");
   outputstream41P=dsGetOutputPortByName("outputstream41P");
   inputstream40P=dsGetInputPortByName("inputstream40P");
   inputstream37P=dsGetInputPortByName("inputstream37P");
   inputstream36P=dsGetInputPortByName("inputstream36P");
   inputstream35P=dsGetInputPortByName("inputstream35P");
   outputstream33P=dsGetOutputPortByName("outputstream33P");
   inputstream32P=dsGetInputPortByName("inputstream32P");
   outputstream30P=dsGetOutputPortByName("outputstream30P");
   inputstream28P=dsGetInputPortByName("inputstream28P");
   inputstream23P=dsGetInputPortByName("inputstream23P");
   inputstream14P=dsGetInputPortByName("inputstream14P");
   inputstream9P=dsGetInputPortByName("inputstream9P");
   inputstream4P=dsGetInputPortByName("inputstream4P");
   return 0;
}

int processFilter(void * work, int size)
{
   int support;
   int i;

   int * combinations;
   combinations = (int *)malloc(sizeof(int)*MAXCOMB);
   for(i=0;i<MAXCOMB;i++){
      combinations[i]=0;
   }

   int ** new_candidates;
   new_candidates = (int **)malloc(sizeof(int *)*MAXCAND);
   for( i = 0;i<MAXCAND;i++){
      new_candidates[i] = (int *)malloc(sizeof(int)*MAX_ITEMS);
   }
```

```
    int n_new_cand;
    int ** candidates;
    candidates = (int **)malloc(sizeof(int*)*MAXCAND);
    for( i = 0;i<MAXCAND;i++){
        candidates[i] = (int *)malloc(sizeof(int)*MAX_ITEMS);
    }

    int * cand_freq;
    cand_freq = (int *)malloc(sizeof(int)*MAXCAND);

    int * new_comb;
    new_comb = (int *)malloc(sizeof(int)*MAX_ITEMS);

    int nu_dimentions;
    int j;
    int k;
    int nu_candidates;
    int cardinality;
    int new_cardinality;
    int aux2;

    printf("Processing filter4\n");
    struct timeval tbegin,tend;
    char *instanceTime;
instanceTime = (char *)calloc(MAXLINESIZE,sizeof(char));


    for (j=0; j<MAX_ITEMS; j ++ ) {
        new_comb[j]=0;
    }

    int ___i___ ;
    message_int * _new_message_int_;
    _new_message_int_ = (message_int*)malloc(sizeof(message_int));

    message_int_array * _new_message_int_array_;
    _new_message_int_array_ = (message_int_array*)malloc(sizeof(
        message_int_array));


    //using variable candidates#5
    int  __index_candidates__ = 0 ;
    int * candidates_index ;
```

```
candidates_index  =(int*)malloc(sizeof(int)*MAXCAND);
while ( dsReadBuffer(inputstream14P, (    _new_message_int_array_),
    sizeof( message_int_array))!= EOW ) {
   memcpy(candidates [  __index_candidates__ ] ,
       _new_message_int_array_->value ,sizeof(int)*MAX_ITEMS);

   candidates_index [  __index_candidates__ ] =
       _new_message_int_array_->index ;

   __index_candidates__ ++ ;
}


// Using variable support#2
//support = ([int])((Work*)work)->support;

dsReadBuffer(inputstream23P, ( & nu_dimentions), sizeof(nu_dimentions)
    );
dsReadBuffer(inputstream28P, ( & nu_candidates), sizeof(nu_candidates)
    );
dsReadBuffer(inputstream9P, ( & cardinality), sizeof(cardinality));
gettimeofday(&tbegin,NULL);

while ((dsReadBuffer(inputstream4P, ( & __new_iteration__), sizeof(
    __new_iteration__))!=-2))
{



   dsReadBuffer(inputstream36P, ( & new_cardinality), sizeof(
       new_cardinality));
   dsReadBuffer(inputstream32P, ( & n_new_cand), sizeof(n_new_cand));


   //broadcast receive from all
   //using variable combinations#5
   for (___i___ = 0; ___i___ < nu_candidates;___i___++) {
      dsReadBuffer(inputstream35P, (    _new_message_int_), sizeof(
          message_int)); ;
      combinations [ _new_message_int_->index    ] =  _new_message_int_
          ->value ;
   }
```

```
for(i=0;i<nu_candidates;i++){
    cand_freq[i]=0;
}
for(i=0;i<nu_candidates*dsGetNumWriters(inputstream47P);i++){
    dsReadBuffer(inputstream47P, (  _new_message_int_), sizeof(
        message_int));
    cand_freq [ _new_message_int_->index  ] +=  _new_message_int_->
        value ;
}

for (i=0; (i<(nu_candidates-1))&&(cand_freq[i]>support); i ++ )
{
    for (j=(i+1); ((j<nu_candidates)&&(j!=nu_candidates))&&(
        cand_freq[j]>support); j ++ )
    {
        aux2=0;
        for (k=0; k<nu_dimentions; k ++ )
        {
            new_comb[k]=(candidates[i][k]||candidates[j][k]);
            new_cardinality+=new_comb[k];
            if (new_comb[k])
            {
                aux2=(aux2<<1);
                aux2+=1;
            }
            else
            {
                aux2=(aux2<<1);
            }

        }

        if ((new_cardinality==(cardinality+1))&&(combinations[aux2
            ]==0))
        {
            for (k=0; k<nu_dimentions; k ++ )
            {
                new_candidates[n_new_cand][k]=new_comb[k];
            }

            n_new_cand=(n_new_cand+1);
            combinations[aux2]=1;
        }
```

```
            for (k=0; k<nu_dimentions; k ++ )
            {
                new_comb[k]=0;
            }

            new_cardinality=0;
        }

    }
    dsWriteBuffer(outputstream33P, ( & n_new_cand), sizeof(n_new_cand))
        ;
    for (i=0;i<n_new_cand;i++){
        _new_message_int_array_->index = i ;
        memcpy (_new_message_int_array_->value , new_candidates[i],
            sizeof(int)*MAX_ITEMS) ;

        dsWriteBuffer(outputstream41P, ( _new_message_int_array_),
            sizeof(message_int_array));
    }


    dsReadBuffer(inputstream40P, ( & nu_candidates), sizeof(
        nu_candidates));
    dsReadBuffer(inputstream37P, ( & cardinality), sizeof(cardinality))
        ;
    for(i=0;i<nu_candidates;i++){
        dsReadBuffer(inputstream44P, (    _new_message_int_array_),
            sizeof( message_int_array));
        memcpy(candidates [ i ], _new_message_int_array_->value ,
            sizeof(int)*MAX_ITEMS);
        candidates_index [ i ] = _new_message_int_array_->index ;
    }


}
dsWriteBuffer(outputstream30P, (  combinations), sizeof(int)*MAXCOMB);

gettimeofday(&tend,NULL);
strcpy(instanceTime,elapsed_time(tbegin,tend));

dsWriteBuffer(outputstream52P, ( instanceTime), sizeof(char )*
    MAXLINESIZE);
return 0;
}
```

```
int finalizeFilter(  )
{
   printf("stoping filter4\n");
   return 0;
}
```

B.1.2.10 filter4.h

```
#ifndef FILTER4_H_
#define FILTER4_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER4_H_*/
```

B.1.2.11 filter5.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter5.h"
int __new_iteration__;
InputPortHandler inputstream0P;
InputPortHandler inputstream10P;
InputPortHandler inputstream24P;
InputPortHandler inputstream29P;
InputPortHandler inputstream33P;
OutputPortHandler outputstream37P;
```

```
OutputPortHandler outputstream38P;
OutputPortHandler outputstream39P;
OutputPortHandler outputstream40P;
InputPortHandler inputstream41P;
OutputPortHandler outputstream42P;
OutputPortHandler outputstream43P;
OutputPortHandler outputstream44P;
OutputPortHandler outputstream53P;
OutputPortHandler outputstream55P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter5\n");
   outputstream55P=dsGetOutputPortByName("outputstream55P");
   outputstream53P=dsGetOutputPortByName("outputstream53P");
   outputstream44P=dsGetOutputPortByName("outputstream44P");
   outputstream43P=dsGetOutputPortByName("outputstream43P");
   outputstream42P=dsGetOutputPortByName("outputstream42P");
   inputstream41P=dsGetInputPortByName("inputstream41P");
   outputstream40P=dsGetOutputPortByName("outputstream40P");
   outputstream39P=dsGetOutputPortByName("outputstream39P");
   outputstream38P=dsGetOutputPortByName("outputstream38P");
   outputstream37P=dsGetOutputPortByName("outputstream37P");
   inputstream33P=dsGetInputPortByName("inputstream33P");
   inputstream29P=dsGetInputPortByName("inputstream29P");
   inputstream24P=dsGetInputPortByName("inputstream24P");
   inputstream10P=dsGetInputPortByName("inputstream10P");
   inputstream0P=dsGetInputPortByName("inputstream0P");
   return 0;
}

int processFilter(void * work, int size)
{

   int i;
   int ** new_candidates;
   new_candidates = (int **)malloc(sizeof(int *)*MAXCAND);
   for( i = 0;i<MAXCAND;i++){
      new_candidates[i] = (int *)malloc(sizeof(int)*MAX_ITEMS);
   }

   int n_new_cand;
   int ** candidates;
   candidates = (int **)malloc(sizeof(int*)*MAXCAND);
   for( i = 0;i<MAXCAND;i++){
```

```
    candidates[i] = (int *)malloc(sizeof(int)*MAX_ITEMS);
}


message_int_array * _new_message_int_array_;
_new_message_int_array_ = (message_int_array*)malloc(sizeof(
    message_int_array));


int nu_dimentions;
int k;
int nu_candidates;
int cardinality;

printf("Processing filter5\n");
struct timeval tbegin,tend;
char *instanceTime;
instanceTime = (char *)calloc(MAXLINESIZE,sizeof(char));


dsReadBuffer(inputstream24P, (&nu_dimentions), sizeof(nu_dimentions)
    );
dsReadBuffer(inputstream29P, (&nu_candidates), sizeof(nu_candidates)
    );
dsReadBuffer(inputstream10P, (&cardinality), sizeof(cardinality));
gettimeofday(&tbegin,NULL);

while ((dsReadBuffer(inputstream0P, (&__new_iteration__), sizeof(
    __new_iteration__))!=-2))
{
    dsReadBuffer(inputstream33P, (&n_new_cand), sizeof(n_new_cand));
    for(i=0;i<n_new_cand;i++){
        dsReadBuffer(inputstream41P, (    _new_message_int_array_),
            sizeof( message_int_array));
        memcpy(new_candidates [ i ], _new_message_int_array_->value ,
            sizeof(int)*MAX_ITEMS);

    }

    {

        nu_candidates=0;
        for (i=0; i<n_new_cand; i ++ )
        {
```

```
            for (k=0; k<nu_dimentions; k ++ )
            {
                candidates[i][k]=new_candidates[i][k];
            }

            nu_candidates=(nu_candidates+1);
        }

        cardinality=(cardinality+1);
        dsWriteBuffer(outputstream55P, ( & nu_candidates), sizeof(
            nu_candidates));
        dsWriteBuffer(outputstream40P, ( & nu_candidates), sizeof(
            nu_candidates));
        dsWriteBuffer(outputstream39P, ( & nu_candidates), sizeof(
            nu_candidates));
        dsWriteBuffer(outputstream38P, ( & nu_candidates), sizeof(
            nu_candidates));


        dsWriteBuffer(outputstream37P, ( & cardinality), sizeof(
            cardinality));
        for(i=0;i<nu_candidates;i++){
            _new_message_int_array_->index = i ;
            memcpy (_new_message_int_array_->value , candidates[i],sizeof
                (int)*MAX_ITEMS) ;
            dsWriteBuffer(outputstream44P, (    _new_message_int_array_),
                sizeof( message_int_array)); ;
            dsWriteBuffer(outputstream43P, (    _new_message_int_array_),
                sizeof( message_int_array)); ;
            dsWriteBuffer(outputstream42P, (    _new_message_int_array_),
                sizeof( message_int_array)); ;
        }


    }

}

gettimeofday(&tend,NULL);
strcpy(instanceTime,elapsed_time(tbegin,tend));

dsWriteBuffer(outputstream53P, ( instanceTime), sizeof(char )*
    MAXLINESIZE);
return 0;
```

```
}

int finalizeFilter( )
{
   printf("stoping filter5\n");
   return 0;
}
```

B.1.2.12 filter5.h

```
#ifndef FILTER5_H_
#define FILTER5_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER5_H_*/
```

B.1.2.13 filter6.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>
#include <math.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter6.h"
int __new_iteration__;
OutputPortHandler outputstream6P;
OutputPortHandler outputstream9P;
OutputPortHandler outputstream10P;
OutputPortHandler outputstream11P;
```

```
OutputPortHandler outputstream12P;
OutputPortHandler outputstream13P;
OutputPortHandler outputstream14P;
OutputPortHandler outputstream18P;
OutputPortHandler outputstream19P;
OutputPortHandler outputstream21P;
OutputPortHandler outputstream22P;
OutputPortHandler outputstream23P;
OutputPortHandler outputstream24P;
OutputPortHandler outputstream25P;
OutputPortHandler outputstream26P;
OutputPortHandler outputstream27P;
OutputPortHandler outputstream28P;
OutputPortHandler outputstream29P;
OutputPortHandler outputstream54P;
OutputPortHandler outputstream5P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter6\n");
   outputstream54P=dsGetOutputPortByName("outputstream54P");
   outputstream29P=dsGetOutputPortByName("outputstream29P");
   outputstream28P=dsGetOutputPortByName("outputstream28P");
   outputstream27P=dsGetOutputPortByName("outputstream27P");
   outputstream26P=dsGetOutputPortByName("outputstream26P");
   outputstream25P=dsGetOutputPortByName("outputstream25P");
   outputstream24P=dsGetOutputPortByName("outputstream24P");
   outputstream23P=dsGetOutputPortByName("outputstream23P");
   outputstream22P=dsGetOutputPortByName("outputstream22P");
   outputstream21P=dsGetOutputPortByName("outputstream21P");
   outputstream19P=dsGetOutputPortByName("outputstream19P");
   outputstream18P=dsGetOutputPortByName("outputstream18P");
   outputstream5P=dsGetOutputPortByName("outputstream5P");
   outputstream14P=dsGetOutputPortByName("outputstream14P");
   outputstream13P=dsGetOutputPortByName("outputstream13P");
   outputstream12P=dsGetOutputPortByName("outputstream12P");
   outputstream11P=dsGetOutputPortByName("outputstream11P");
   outputstream10P=dsGetOutputPortByName("outputstream10P");
   outputstream9P=dsGetOutputPortByName("outputstream9P");
   outputstream6P=dsGetOutputPortByName("outputstream6P");
   return 0;
}

int processFilter(void * work, int size)
{
```

```
char *dbFileName;
dbFileName = (char *)malloc(sizeof(char)*MAXLINESIZE);


int i;
int ** itemsets;
itemsets = (int **)malloc(sizeof(int*)*MAX_ITEMSETS);
for( i = 0;i<MAX_ITEMSETS;i++){
   itemsets[i] = (int *)malloc(sizeof(int)*MAX_ITEMS);
}

int * combinations;
combinations = (int *)malloc(sizeof(int)*MAXCOMB);


int * comb_freq;
comb_freq = (int *)malloc(sizeof(int)*MAXCOMB);

int ** candidates;
candidates = (int **)malloc(sizeof(int*)*MAXCAND);
for( i = 0;i<MAXCAND;i++){
   candidates[i] = (int *)malloc(sizeof(int)*MAX_ITEMS);
}


int * new_comb;
new_comb = (int *)malloc(sizeof(int)*MAX_ITEMS);

int nu_itemsets;
int nu_dimentions;
int temp_dimentions;
int j;
int l;
int nu_candidates;
int cardinality;
message_int * _new_message_int_;
_new_message_int_ = (message_int*)malloc(sizeof(message_int));
message_int_array * _new_message_int_array_;
_new_message_int_array_ = (message_int_array*)malloc(sizeof(
    message_int_array));


FILE * Arq;
char * line ;
```

```
line= (char *)malloc(sizeof(char)*MAXLINESIZE);

char * token;

printf("Processing filter6\n");
struct timeval tbegin,tend;
char *instanceTime;
instanceTime = (char *)calloc(MAXLINESIZE,sizeof(char));

gettimeofday(&tbegin,NULL);
// Using variable dbFileName#2
strcpy (dbFileName , (char*)((Work*)work)->dbFileName);
{
   if ((Arq=fopen(dbFileName, "r+")))
   {
      nu_itemsets=0;
      nu_dimentions=0;
      temp_dimentions=0;
      for (i=0;  ! feof(Arq); i=0) {
         line[0]='\0';
         fgets(line, MAXLINESIZE, Arq);
         l=strlen(line);
         //remove \n char
         if (line[(l-1)]=='\n')
         {
            l--;
         }

         line[l]='\0';
         if (l>0)
         {
            token=strtok(line," ");

            for (i=0; token!=((void *)0); i=0)
            {
               itemsets[nu_itemsets][temp_dimentions]=atoi(token);
               temp_dimentions=(temp_dimentions+1);
               token=strtok(((void *)0)," ");

            }


            nu_dimentions=temp_dimentions;
            nu_itemsets=(nu_itemsets+1);
```

```
            temp_dimentions=0;
        }

    }

}
else
{
    printf("Cannot open dbFile: %s\n", dbFileName);
    exit(0);
}
fclose(Arq);
for(i=0;i<nu_itemsets;i++){
    _new_message_int_array_->index = i ;
    memcpy (_new_message_int_array_->value , itemsets[i],sizeof(int)
        *MAX_ITEMS) ;
    dsWriteBuffer(outputstream11P, (    _new_message_int_array_),
        sizeof( message_int_array)); ;
}

dsCloseOutputPort ( outputstream11P ) ;

dsWriteBuffer(outputstream25P, ( & nu_dimentions), sizeof(
    nu_dimentions));
dsWriteBuffer(outputstream24P, ( & nu_dimentions), sizeof(
    nu_dimentions));
dsWriteBuffer(outputstream23P, ( & nu_dimentions), sizeof(
    nu_dimentions));
dsWriteBuffer(outputstream22P, ( & nu_dimentions), sizeof(
    nu_dimentions));
dsWriteBuffer(outputstream21P, ( & nu_dimentions), sizeof(
    nu_dimentions));
dsWriteBuffer(outputstream18P, ( & nu_itemsets), sizeof(nu_itemsets
    ));


if (pow(2, nu_dimentions)>MAXCOMB)
{instanceTime = (char *)calloc(MAXLINESIZE,sizeof(char));

    printf("MUST increase MAXCOMB to run\n");
    exit(0);
}
```

```
if (nu_itemsets>MAX_ITEMSETS)
{
   printf("MUST increase MAX_ITEMSETS to run\n");
   exit(0);
}

for (i=0; i<MAXCOMB; i ++ )
{

   for (j=0; j<MAX_ITEMS; j ++ )
   {
      candidates[i][j]=0;
   }


   combinations[i]=0;

   _new_message_int_->index = i ;
   _new_message_int_->value = combinations[i] ;
   dsWriteBuffer(outputstream6P, (    _new_message_int_), sizeof(
       message_int)); ;

   comb_freq[i]=0;

   _new_message_int_->index = i ;
   _new_message_int_->value = comb_freq[i] ;
   dsWriteBuffer(outputstream19P, (    _new_message_int_), sizeof(
       message_int)); ;

}

dsCloseOutputPort ( outputstream19P ) ;
dsCloseOutputPort ( outputstream6P ) ;

nu_candidates=0;
for (i=0; i<nu_dimentions; i ++ )
{
   candidates[i][i]=1;
   nu_candidates=(nu_candidates+1);
}
dsWriteBuffer(outputstream29P, ( & nu_candidates), sizeof(
   nu_candidates));
dsWriteBuffer(outputstream28P, ( & nu_candidates), sizeof(
   nu_candidates));
```

```
    dsWriteBuffer(outputstream27P, ( & nu_candidates), sizeof(
        nu_candidates));
    dsWriteBuffer(outputstream26P, ( & nu_candidates), sizeof(
        nu_candidates));
    dsWriteBuffer(outputstream5P, ( & nu_candidates), sizeof(
        nu_candidates));

    cardinality=1;
    dsWriteBuffer(outputstream10P, ( & cardinality), sizeof(cardinality
        ));
    dsWriteBuffer(outputstream9P, ( & cardinality), sizeof(cardinality)
        );
    for (i=0; i<nu_candidates; i ++ ){
        _new_message_int_array_->index = i ;
       memcpy (_new_message_int_array_->value , candidates[i],sizeof(
           int)*MAX_ITEMS) ;
        dsWriteBuffer(outputstream14P, (    _new_message_int_array_),
           sizeof( message_int_array)); ;
        dsWriteBuffer(outputstream13P, (    _new_message_int_array_),
           sizeof( message_int_array)); ;
        dsWriteBuffer(outputstream12P, (    _new_message_int_array_),
           sizeof( message_int_array)); ;
    }


}







dsCloseOutputPort ( outputstream12P ) ;

dsCloseOutputPort ( outputstream13P ) ;

dsCloseOutputPort ( outputstream14P ) ;







gettimeofday(&tend,NULL);
```

```
    strcpy(instanceTime,elapsed_time(tbegin,tend));

    dsWriteBuffer(outputstream54P, (instanceTime), sizeof(char)*
        MAXLINESIZE);

    return 0;
}

int finalizeFilter( )
{
    printf("stoping filter6\n");
    return 0;
}
```

B.1.2.14 filter6.h

```
#ifndef FILTER6_H_
#define FILTER6_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER6_H_*/
```

B.1.2.15 filter7.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>
#include <math.h>

#include "work.h"
#include "util.h"
#include "messages.h"
```

```
#include "filter7.h"
int __new_iteration__;
InputPortHandler inputstream25P;
InputPortHandler inputstream30P;
InputPortHandler inputstream31P;
InputPortHandler inputstream48P;
InputPortHandler inputstream49P;
InputPortHandler inputstream50P;
InputPortHandler inputstream51P;
InputPortHandler inputstream52P;
InputPortHandler inputstream53P;
InputPortHandler inputstream54P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter7\n");
   inputstream54P=dsGetInputPortByName("inputstream54P");
   inputstream53P=dsGetInputPortByName("inputstream53P");
   inputstream52P=dsGetInputPortByName("inputstream52P");
   inputstream51P=dsGetInputPortByName("inputstream51P");
   inputstream50P=dsGetInputPortByName("inputstream50P");
   inputstream49P=dsGetInputPortByName("inputstream49P");
   inputstream48P=dsGetInputPortByName("inputstream48P");
   inputstream31P=dsGetInputPortByName("inputstream31P");
   inputstream30P=dsGetInputPortByName("inputstream30P");
   inputstream25P=dsGetInputPortByName("inputstream25P");
   return 0;
}

int processFilter(void * work, int size)
{
   int support;

   int * combinations;
   combinations = (int *)malloc(sizeof(int)*MAXCOMB);

   int * comb_freq;
   comb_freq = (int *)malloc(sizeof(int)*MAXCOMB);

   int nu_dimentions;
   int j;
   int i;
   int nu_candidates;

   printf("Processing filter7\n");
```

```
struct timeval tbegin, tend;
char *instanceTime;
instanceTime = (char *)calloc(MAXLINESIZE,sizeof(char));


char outputFileName[256];
FILE * Arq;
InputPortHandler *timeStreams[7 ];
gettimeofday(&tbegin,NULL);

dsReadBuffer(inputstream25P, ( & nu_dimentions), sizeof(nu_dimentions)
    );
//using variable comb_freq#3
message_int * _new_message_int_;
_new_message_int_ = (message_int*)malloc(sizeof(message_int));
while ( dsReadBuffer(inputstream31P, (    _new_message_int_), sizeof(
    message_int))!= EOW ) {
    comb_freq [ _new_message_int_->index ] =  _new_message_int_->value
        ;
}

dsReadBuffer(inputstream30P, (   combinations), sizeof(int)*MAXCOMB);


// Using variable support#2
support = (int)((Work*)work)->support;
{
    nu_candidates=pow(2, nu_dimentions);
    for (i=0; i<nu_candidates; i ++ )
    {
        if (combinations[i])
        {
            if (comb_freq[i]>support)
            {
                printf("comb:%d freq:%d\n", i, comb_freq[i]);
            }

        }

    }

}

sprintf (outputFileName,"%s_%d","outputWriterFilter",(int)time(NULL));
```

```
    Arq=fopen(outputFileName,"w+");
    timeStreams[0] = & inputstream48P ;
    timeStreams[1] = & inputstream49P ;
    timeStreams[2] = & inputstream50P ;
    timeStreams[3] = & inputstream51P ;
    timeStreams[4] = & inputstream52P ;
    timeStreams[5] = & inputstream53P ;
    timeStreams[6] = & inputstream54P ;
    dsReadBuffer(*timeStreams[6], ( instanceTime ), sizeof(char)*
        MAXLINESIZE);
    fprintf(Arq,"Filter %d read TIME: %s\n",6,instanceTime);
    for (i=0;i<6;i++){
        for (j=0;j<dsGetNumWriters(*timeStreams[i]);j++){
            dsReadBuffer(*timeStreams[i], ( instanceTime ), sizeof(char)*
                MAXLINESIZE);
            fprintf(Arq,"Filter %d instance %d TIME: %s\n",i,j,instanceTime)
                ;
        }
    }
    gettimeofday(&tend,NULL);
    fprintf(Arq,"Filter 7 TIME: %s\n",elapsed_time(tbegin,tend));
    fclose(Arq);
    return 0;

}

int finalizeFilter( )
{
    printf("stoping filter7\n");
    return 0;
}
```

B.1.2.16 filter7.h

```
#ifndef FILTER7_H_
#define FILTER7_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();
```

```
#endif /*FILTER7_H_*/
```

B.1.2.17 itemsets.c

```
#include <stdio.h>

#include <stdlib.h>


#include <string.h>

#include <math.h>

#include "work.h"
#include "void.h"
#include "util.h"
int main(int argc, char *  * argv)
{
    int support;
    char confFile[] = "./conf.xml";
    char *dbFileName;
    dbFileName = (char *)malloc(sizeof(char)*MAXLINESIZE);
    Work *work = (Work *)malloc(sizeof(Work));

    if (argc<5)
    {
        printf("Usage: %s <database_file> <support> <nu_itemsets>
            nu_dimentions\n", argv[0]);
        exit(0);
    }

    strcpy(dbFileName, argv[1]);
    support=atoi(argv[2]);

    /*
        Statement removed because pragma READ annotation found
        */
    work->support=support;
    work->nu_itemsets=atoi(argv[3]);
    work->nu_dimentions=atoi(argv[4]);
    strcpy(work->dbFileName,dbFileName);
```

```
    Layout *systemLayout = initDs(confFile, argc, argv);
    appendWork(systemLayout, (void *)work, sizeof(Work));
    finalizeDs(systemLayout);
    /*
#WHILE LOOP REMOVED - FILTERS INSERTED:  0 1 2 3 4 5
*/
    /*
        Statement removed because pragma WRITE annotation found
        */
    return 0;
}
```

B.1.2.18 labelFunc.c

```
int hash(char * label, int image)
{
    int dest;
    int * aux;
    aux=((int * )( & label[0]));
    dest=(( * aux)%image);
    return dest;
}

void getLabel(void * msg, int size, char label[])
{
    int * aux;
    aux=((int * )( & label[0]));
    ( * aux)=( * ((int * )msg));
}
```

B.1.2.19 messages.h

```
#include "work.h"
struct message_int_array_t
{
    int index;
    int value[MAX_ITEMS];
};

typedef struct message_int_array_t message_int_array, *
    message_int_array_ptr;
```

```
struct message_int_t
{
   int index;
   int value;
};

typedef struct message_int_t message_int, * message_int_ptr;
```

B.1.2.20 work.h

```
#ifndef _WORK_H_
#define _WORK_H_
#define MAX_ITEMSETS 11000
#define MAX_ITEMS 21
#define MAXLINESIZE 1024
#define MAXCOMB 1048576
#define MAXCAND 1048576

struct _Work
{
   int support;
   char dbFileName[MAXLINESIZE];
   int nu_itemsets;
   int nu_dimentions;
};

typedef struct _Work Work, * Work_ptr;
#endif
```

B.1.2.21 conf.xml

```
<config>
   <hostdec>
      <host name="uzi01" mem="2048">
         <resource name="1"/>
         </host>
      <host name="uzi02" mem="2048">
         <resource name="2"/>
         </host>
      <host name="uzi03" mem="2048">
         <resource name="3"/>
         </host>
```

```
<host name="uzi05" mem="2048">
   <resource name="4"/>
   </host>
<host name="uzi06" mem="2048">
   <resource name="5"/>
   </host>
<host name="uzi08" mem="2048">
   <resource name="6"/>
   </host>
<host name="uzi09" mem="2048">
   <resource name="7"/>
   </host>
<host name="uzi10" mem="2048">
   <resource name="8"/>
   </host>
<host name="uzi11" mem="2048">
   <resource name="9"/>
   </host>
<host name="uzi12" mem="2048">
   <resource name="10"/>
   </host>
      <host name="uzi14" mem="2048">
   <resource name="11"/>
   </host>
<host name="uzi17" mem="2048">
   <resource name="12"/>
   </host>
<host name="uzi18" mem="2048">
   <resource name="13"/>
   </host>
<host name="uzi19" mem="2048">
   <resource name="14"/>
   </host>
<host name="uzi20" mem="2048">
   <resource name="15"/>
   </host>
<host name="uzi21" mem="2048">
   <resource name="16"/>
   </host>
<host name="uzi22" mem="2048">
   <resource name="17"/>
   </host>
<host name="uzi23" mem="2048">
   <resource name="18"/>
```

```
        </host>
    <host name="uzi24" mem="2048">
        <resource name="19"/>
        </host>
    <host name="uzi25" mem="2048">
        <resource name="20"/>
        </host>
    <host name="uzi28" mem="2048">
        <resource name="21"/>
        </host>
    <host name="uzi32" mem="2048">
        <resource name="22"/>
        </host>
    <host name="uzi33" mem="2048">
        <resource name="23"/>
        </host>
    <host name="uzi35" mem="2048">
        <resource name="24"/>
        </host>
</hostdec>

<placement>
    <filter name="filter0" libname="filter0.so" instances="1">
    <instance demands="2" numinstances="1" />
    </filter>
    <filter name="filter2" libname="filter2.so" instances="1">
    <instance demands="2" numinstances="1" />
    </filter>
    <filter name="filter4" libname="filter4.so" instances="1">
    <instance demands="2" numinstances="1" />
    </filter>
    <filter name="filter5" libname="filter5.so" instances="1">
    <instance demands="3" numinstances="1" />
    </filter>
    <filter name="filter6" libname="filter6.so" instances="1">
    <instance demands="1" numinstances="1" />
    </filter>
    <filter name="filter7" libname="filter7.so" instances="1">
    <instance demands="4" numinstances="1" />
    </filter>
<filter name="filter1" libname="filter1.so" instances="1">
    <instance demands="5" numinstances="1" />
</filter>
```

```
<filter name="filter3" libname="filter3.so" instances="4">
   <instance demands="6" numinstances="1" />
   <instance demands="7" numinstances="1" />
   <instance demands="8" numinstances="1" />
   <instance demands="9" numinstances="1" />
</filter>

</placement>
<layout>
   <stream>
      <from filter="filter0" port="outputstream4P" policy="broadcast"
         />
      <to filter="filter4" port="inputstream4P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream48P" policy="broadcast"
          />
      <to filter="filter7" port="inputstream48P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream1P" policy="broadcast"
         />
      <to filter="filter2" port="inputstream1P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream0P" policy="broadcast"
         />
      <to filter="filter5" port="inputstream0P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream3P" policy="broadcast"
         />
      <to filter="filter1" port="inputstream3P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream2P" policy="broadcast"
         />
      <to filter="filter3" port="inputstream2P"/>
   </stream>
   <stream>
      <from filter="filter1" port="outputstream49P" policy="broadcast"
          />
      <to filter="filter7" port="inputstream49P"/>
   </stream>
```

```
<stream>
   <from filter="filter2" port="outputstream50P" policy="broadcast"
       />
   <to filter="filter7" port="inputstream50P"/>
</stream>
<stream>
   <from filter="filter2" port="outputstream36P" policy="broadcast"
       />
   <to filter="filter4" port="inputstream36P"/>
</stream>
<stream>
   <from filter="filter2" port="outputstream32P" policy="broadcast"
       />
   <to filter="filter4" port="inputstream32P"/>
</stream>
<stream>
   <from filter="filter3" port="outputstream51P" policy="broadcast"
       />
   <to filter="filter7" port="inputstream51P"/>
</stream>
<stream>
   <from filter="filter3" port="outputstream35P" policy="
      labeled_stream"   policyLib="labelFunc.so"   />
   <to filter="filter4" port="inputstream35P"/>
</stream>
<stream>
   <from filter="filter3" port="outputstream31P" policy="
      round_robin" />
   <to filter="filter7" port="inputstream31P"/>
</stream>
<stream>
   <from filter="filter4" port="outputstream52P" policy="broadcast"
       />
   <to filter="filter7" port="inputstream52P"/>
</stream>
<stream>
   <from filter="filter4" port="outputstream33P" policy="broadcast"
       />
   <to filter="filter5" port="inputstream33P"/>
</stream>
<stream>
   <from filter="filter4" port="outputstream30P" policy="
      round_robin" />
   <to filter="filter7" port="inputstream30P"/>
```

```
</stream>
<stream>
   <from filter="filter4" port="outputstream41P" policy="
      labeled_stream"  policyLib="labelFunc.so"  />
   <to filter="filter5" port="inputstream41P"/>
</stream>
<stream>
   <from filter="filter5" port="outputstream53P" policy="broadcast"
       />
   <to filter="filter7" port="inputstream53P"/>
</stream>
<stream>
   <from filter="filter5" port="outputstream39P" policy="broadcast"
       />
   <to filter="filter3" port="inputstream39P"/>
</stream>
<stream>
   <from filter="filter5" port="outputstream37P" policy="broadcast"
       />
   <to filter="filter4" port="inputstream37P"/>
</stream>
<stream>
   <from filter="filter5" port="outputstream38P" policy="broadcast"
       />
   <to filter="filter1" port="inputstream38P"/>
</stream>
<stream>
   <from filter="filter5" port="outputstream43P" policy="
      labeled_stream"  policyLib="labelFunc.so"  />
   <to filter="filter3" port="inputstream43P"/>
</stream>
<stream>
   <from filter="filter5" port="outputstream42P" policy="broadcast"
        />
   <to filter="filter1" port="inputstream42P"/>
</stream>
<stream>
   <from filter="filter5" port="outputstream40P" policy="broadcast"
       />
   <to filter="filter4" port="inputstream40P"/>
</stream>
<stream>
   <from filter="filter1" port="outputstream47P" policy="
      labeled_stream"  policyLib="labelFunc.so"  />
```

```
    <to filter="filter4" port="inputstream47P"/>
  </stream>
  <stream>
    <from filter="filter5" port="outputstream44P" policy="
        labeled_stream" policyLib="labelFunc.so" />
    <to filter="filter4" port="inputstream44P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream13P" policy="
        labeled_stream" policyLib="labelFunc.so" />
    <to filter="filter3" port="inputstream13P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream54P" policy="broadcast"
        />
    <to filter="filter7" port="inputstream54P"/>
  </stream>
  <stream>
    <from filter="filter5" port="outputstream55P" policy="broadcast"
        />
    <to filter="filter0" port="inputstream55P"/>
  </stream>

  <stream>
    <from filter="filter6" port="outputstream14P" policy="broadcast"
        />
    <to filter="filter4" port="inputstream14P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream11P" policy="
        round_robin" />
    <to filter="filter1" port="inputstream11P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream12P" policy="broadcast"
        />
    <to filter="filter1" port="inputstream12P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream17P" policy="broadcast"
        />
    <to filter="filter4" port="inputstream17P"/>
  </stream>
  <stream>
```

```
    <from filter="filter6" port="outputstream18P" policy="broadcast"
        />
    <to filter="filter1" port="inputstream18P"/>
</stream>
<stream>
    <from filter="filter6" port="outputstream19P" policy="broadcast"
        />
    <to filter="filter3" port="inputstream19P"/>
</stream>
<stream>
    <from filter="filter6" port="outputstream21P" policy="broadcast"
        />
    <to filter="filter1" port="inputstream21P"/>
</stream>
<stream>
    <from filter="filter6" port="outputstream26P" policy="broadcast"
        />
    <to filter="filter1" port="inputstream26P"/>
</stream>
<stream>
    <from filter="filter6" port="outputstream5P" policy="broadcast"
       />
    <to filter="filter0" port="inputstream5P"/>
</stream>

<stream>
    <from filter="filter6" port="outputstream27P" policy="broadcast"
        />
    <to filter="filter3" port="inputstream27P"/>
</stream>
<stream>
    <from filter="filter6" port="outputstream28P" policy="broadcast"
        />
    <to filter="filter4" port="inputstream28P"/>
</stream>
<stream>
    <from filter="filter6" port="outputstream29P" policy="broadcast"
        />
    <to filter="filter5" port="inputstream29P"/>
</stream>
<stream>
    <from filter="filter6" port="outputstream22P" policy="broadcast"
        />
    <to filter="filter3" port="inputstream22P"/>
```

```
      </stream>
      <stream>
         <from filter="filter6" port="outputstream23P" policy="broadcast"
             />
         <to filter="filter4" port="inputstream23P"/>
      </stream>
      <stream>
         <from filter="filter6" port="outputstream24P" policy="broadcast"
             />
         <to filter="filter5" port="inputstream24P"/>
      </stream>
      <stream>
         <from filter="filter6" port="outputstream25P" policy="broadcast"
             />
         <to filter="filter7" port="inputstream25P"/>
      </stream>
      <stream>
         <from filter="filter6" port="outputstream9P" policy="broadcast"
            />
         <to filter="filter4" port="inputstream9P"/>
      </stream>
      <stream>
         <from filter="filter6" port="outputstream7P" policy="broadcast"
            />
         <to filter="filter7" port="inputstream7P"/>
      </stream>
      <stream>
         <from filter="filter6" port="outputstream6P" policy="broadcast"
            />
         <to filter="filter3" port="inputstream6P"/>
      </stream>
      <stream>
         <from filter="filter6" port="outputstream10P" policy="broadcast"
             />
         <to filter="filter5" port="inputstream10P"/>
      </stream>
      <stream>
         <from filter="filter1" port="outputstream56P" policy="
            labeled_stream"  policyLib="labelFunc.so"  />
         <to filter="filter3" port="inputstream56P"/>
      </stream>
   </layout>
</config>
```

B.2 K-means

B.2.1 Entrada

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "estruturas-kmeans_vector.h"
#include <math.h>

int main (  int argc, char *argv[]){
   // Arquivo
   FILE     *Arq;
   char     nome_arquivo[100];
   char     linha[50];

   // Numero de particoes
   int      nu_particoes ;
   // Lista de centros das particoes - id do ponto
   // t_Item lista_part_centrs[NUPARTICOES];
   int      lista_part_centros_ativo[NUPARTICOES];
   int      lista_part_centros_id_item[NUPARTICOES];
   char      lista_part_centros_label[NUPARTICOES][10];
   float      lista_part_centros_x[NUPARTICOES];
   float      lista_part_centros_y[NUPARTICOES];

   float      lista_part_centros_z[NUPARTICOES];
   float      lista_part_centros_w[NUPARTICOES];

   // Lista de itens das particoes - 0 ou 1
   int      lista_part_itens[NUPARTICOES][NUITENS];

   // Lista de distancias do ponto ao centro de cada particao
   float    lista_distancia[NUPARTICOES];

   // Pontos (x,y)
   //t_Item  lista_pontos[NUITENS];
   int lista_pontos_ativo[NUITENS];
   int lista_pontos_id_item[NUITENS];
   char lista_pontos_label[NUITENS][10];
   float lista_pontos_x[NUITENS];
   float lista_pontos_y[NUITENS];
   float lista_pontos_z[NUITENS];
```

```
float lista_pontos_w[NUITENS];
int      nu_pontos;

int m;
int  i;
int  j;
// Temp, contadores
int  l;

int k;
int  t;

float  x;
float y;
float  z;
float w;

float  x1;
float y1;
float  x2;
float y2;

// Str Temp
char     str_tmp[30];

int nu_max_iteracao;
int menor;
int itens;
int ini;

float soma_x;
float soma_y;
float media_x;
float media_y;

int  iteracao;

//iniciando variaveis
Arq=NULL;
nu_particoes=0;
nu_pontos=0;
m=0;
i=0;
j=0;
```

```
l=0;
k=0;
t=0;

x=0.0;
y=0.0;
z=0.0;
w=0.0;

x1=0.0;
y1=0.0;
x2=0.0;
y2=0.0;

nu_max_iteracao=0;
menor=0;
itens=0;
ini=0;

soma_x=0.0;
soma_y=0.0;
media_x=0.0;
media_y=0.0;

iteracao=0;
lista_part_centros_ativo[0]=0;
lista_part_centros_id_item[0]=0;
lista_part_centros_label[0][0]=0;
lista_part_centros_x[0]=0;
lista_part_centros_y[0]=0;
lista_part_centros_z[0]=0;
lista_part_centros_w[0]=0;

lista_part_itens[0][0]=0;

lista_distancia[0]=0;

lista_pontos_ativo[0]=0;
lista_pontos_id_item[0]=0;
lista_pontos_label[0][0]=0;
lista_pontos_x[0]=0;
lista_pontos_y[0]=0;
lista_pontos_z[0]=0;
lista_pontos_w[0]=0;
```

```
str_tmp[0]=0;
linha[0]=0;


// ------------------------------------------------------------------
// Pega parametros

if(argc != 4){
   printf("Executar:  ./a.out <nome arquivo> <# de particoes><#
      interacoes> \n");
   exit(0);
}
// passando argumento para inteiro
sprintf(nome_arquivo, "%s", argv[1]);
sscanf(argv[2], "%d", &nu_particoes) ;
sscanf(argv[3], "%d", &nu_max_iteracao) ;

// ------------------------------------------------------------------
if (SAIDA){ printf ("\n --- Início : Arq (%s) - # Particoes (%d) - Int
    (%d)--", nome_arquivo, nu_particoes, nu_max_iteracao);
}

// ------------------ Inicia Variavies ------------------------------
nu_pontos = 0;

for (i=0 ; i<NUPARTICOES ; i++ ) {
   lista_part_centros_x[i] = 0;
   lista_part_centros_y[i] = 0;
   lista_distancia[i]=0;
   for (j=0 ; j<NUITENS ; j++ ) {
      lista_part_itens[i][j] = 0;
   }
}

// ****************************************
//   Inicializa pontos  = arquivo pontos;
//   Inicializa centros = aleatorio;
// ****************************************

// ------------------ Abre o arquivo --------------------------------
// Primeira linha contem os itens ...
if ( (Arq = fopen(nome_arquivo,"r+") ) )
{
   if (SAIDA) printf ("\n --- Ler arquivo - pontos ---");
```

```
nu_pontos = 0;

// Lendo as TUPLAS
for (m=0; !(feof(Arq));m++ )  {

   linha[0] = '\0';
   fgets(linha, 40, Arq);
   l = strlen(linha);
   l --;
   linha[l] = '\0';

   // Se linha tem conteudo
   if (l>0) {
      if (LEITURA) {
         printf ("\n -- Linha (%d):\t(%s) (%d)", nu_pontos, linha,
            l);
      }

      // Lendo dados da Linha
      ini = 0;
      for (i=0 ; i<(NUDIMESAO+1) ; i++) {

         bzero(str_tmp, 10) ;
         // Identifica Item
         for (j=ini ; j<l ; j++) {

            if ( (linha[j] == ' ' ) || (j == (l-1)) ) {
               k = j - ini;
               strncpy(str_tmp,&(linha[ini]), k);
               ini = j+1;
               // Para sair do loop
               j = l+10;
            }
         }

         // Ajuste para atoi
         t = strlen(str_tmp);
         str_tmp[t] = '\0' ;

         // Pega Label
         if (i == 0){
            lista_pontos_ativo[nu_pontos] = 1;
            lista_pontos_id_item[nu_pontos] = nu_pontos;
```

```
                strcpy(lista_pontos_label[nu_pontos],str_tmp);
                // Pega X
            } else {
                if (i == 1) {
                    x = ( atoi(str_tmp) ) ;
                    lista_pontos_x[nu_pontos] = x;
                    // Pega Y
                } else{
                    if (i == 2) {
                        y = ( atoi(str_tmp) ) ;
                        lista_pontos_y[nu_pontos] = y;
                        // Pega Z
                    } else {
                        if (i == 3){
                            z = ( atoi(str_tmp) ) ;
                            lista_pontos_z[nu_pontos] = z;
                            // Pega W
                        } else {
                            if (i == 4){
                                w = ( atoi(str_tmp) ) ;
                                lista_pontos_w[nu_pontos] = w;
                            }
                        }
                    }
                }
            }

        }
        // End For i

        // Incrementa
        nu_pontos =nu_pontos+1;

    }
} // End While    (trocado para for)

fclose(Arq);

for (i=0 ; i<nu_pontos ; i++) {
    if (SAIDA) {
        printf ("\n --> Ponto: (%s) - (%.2f)(%.2f) ",
            lista_pontos_label[i], lista_pontos_x[i], lista_pontos_y[
            i]);
    }
```

```
    }
    for(i =0;i<nu_particoes;i++){
        //choose random point
        j=i;
        lista_part_centros_x[i] = lista_pontos_x[j] ;
        lista_part_centros_y[i] = lista_pontos_y[j];
        lista_part_itens[i][j] = 1;
    }
}

iteracao = 0;
while (iteracao< nu_max_iteracao) {

    if (SAIDA){
        printf ("\n\n > Centros --- nova rodada ---------- ");
    }
    for (i=0 ; i<nu_particoes ; i++) {
        if (SAIDA) {
            printf ("\n --> (%d) - (%.2f)(%.2f) - Grupo (%d)", i,
                lista_part_centros_x[i], lista_part_centros_y[i], i);
        }
    }

    // *******************************************************
    // 4 - Atribuir cada amostra a uma cluster de acordo com a
    //       medida de similaridade.
    //
    //  Para cada ponto (não centro) : p  {
    //     Para cada Centro  : c {
    //        calcula distancia : d = entre c e p
    //     }
    //     Atribui p ao cluster C de menor distacia entre c e p
    // }
    // *******************************************************

    if (SAIDA) {
        printf ("\n -- Calculando os clusters ----------  ");
    }

    // Para cada ponto : i
    for ( i=0 ; i<nu_pontos ; i++ )
    {
        x2 = lista_pontos_x[i];
        y2 = lista_pontos_y[i];
```

```
// Testa se NAo é centro
// A FAZER !!!!
menor = 0;

// Para cada centro: j
for ( j=0 ; j<nu_particoes ; j++ )
{
   // calcula distancia : d = entre k e i (centro e item)
   x1 = lista_part_centros_x[j];
   y1 = lista_part_centros_y[j];

   lista_distancia[j] = sqrt( ( (x1 - x2)*(x1 - x2) ) + ( (y1 -
      y2)*(y1 - y2) )  );

   if (DEBUG)  {
      printf ("\n ++> Ponto (%s) (%.2f)(%.2f) e (%s) (%.2f)(%.2f
         ) - distancia (%.2f) ", lista_pontos_label[i], x2, y2 ,
          lista_pontos_label[j], x1, y1, lista_distancia[j]);
   }

   // Garda menor distância
   if ( lista_distancia[j] < lista_distancia[menor] ) {
      menor = j;
   }

} // End For j

// Tira ponto de outra particao
for ( j=0 ; j<nu_particoes ; j++ )
{
   lista_part_itens[j][i] = 0;
}

// Inclui ponto na partição onde ele tenha menor
// ditancia ao centro ...
lista_part_itens[menor][i] = 1;

if (DEBUG) {
   printf ("\n +++>> Ponto (%s) - (%.2f)(%.2f) - particao (%d)
      -(%d)  \n", lista_pontos_label[i], lista_pontos_x[i],
      lista_pontos_y[i], menor, lista_part_itens[menor][i]);
}
```

```
} // End For i


// ****************************************
//  5 - Calcular a centróide j
//dos novos clusters
//  (j é a média do cluster j, para j=1..K).
//
//     - Calcula Ponto médio de cada particao - M
//     - Vefifica se é o final
//          Se realizou o numero de iteracaoes
// ****************************************
if (SAIDA) {
   printf ("\n -- Calculando os centroides ----------  ");
}

// Para cada partição: i
for (i=0 ; i<nu_particoes ; i++)
{

   soma_x = 0 ;
   soma_y = 0 ;

   itens = 0;

   // Para os itens desta partição
   for (j=0; j<nu_pontos; j++) {

      if ( lista_part_itens[i][j] == 1 ) {

         soma_x = soma_x +lista_pontos_x[j] ;
         soma_y = soma_y +lista_pontos_y[j] ;

         itens=itens+1;
      }
   }

   // Este é o ponto médio do cluster !!
   media_y = soma_y / itens ;
   media_x = soma_x / itens ;

   if (DEBUG) {
      printf ("\n ++> Ponto M (medio) - (%.2f)(%.2f) - Particao (%d
         )",  media_x, media_y, i);
```

```
            }

            // Novo centro é o menor
            lista_part_centros_x[i] = media_x;
            lista_part_centros_y[i] = media_y;

        } // End For i

        iteracao =iteracao+1;

    }
    // End while (teste)

    // ---------------------------------------
    // -----     Resultado Final -------------

    {
        printf ("\n Resultado ------------ ");

        // Para cada partição: i
        for (i=0 ; i<nu_particoes ; i++)
        {
            printf ("\n particao (%d) - Centro: (%.2f)(%.2f) ", i ,
                lista_part_centros_x[i], lista_part_centros_y[i]);

            // Para cada item: j
            for (j=0 ; j<nu_pontos ; j++)                    {
                if ( lista_part_itens[i][j] == 1) {
                    printf ("\n --> Ponto:(%s) - (%.2f)(%.2f)",
                        lista_pontos_label[j], lista_pontos_x[j],
                        lista_pontos_y[j]);
                }
            }

        }


        // ---------------------------------------------------

        printf ("\n\n ----- FIM ----- \n");
    }
    i=0;
    return i;
} // End Main
```

```
// ————————
```

B.2.2 Saída

B.2.2.1 filter0.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "estruturas-kmeans_vector.h"
#include "work.h"
#include "messages.h"
#include "filter0.h"
#include "util.h"
int __new_iteration__;
OutputPortHandler outputstream0P;
OutputPortHandler outputstream1P;
OutputPortHandler outputstream2P;
OutputPortHandler outputstream3P;
OutputPortHandler outputstream4P;
OutputPortHandler outputstream5P;
OutputPortHandler outputstream6P;
InputPortHandler inputstream30P;
OutputPortHandler outputstream36P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter0\n");
   inputstream30P=dsGetInputPortByName("inputstream30P");
   outputstream6P=dsGetOutputPortByName("outputstream6P");
   outputstream5P=dsGetOutputPortByName("outputstream5P");
   outputstream4P=dsGetOutputPortByName("outputstream4P");
   outputstream3P=dsGetOutputPortByName("outputstream3P");
   outputstream2P=dsGetOutputPortByName("outputstream2P");
   outputstream1P=dsGetOutputPortByName("outputstream1P");
   outputstream0P=dsGetOutputPortByName("outputstream0P");
   outputstream36P=dsGetOutputPortByName("outputstream36P");
   return 0;
```

```
}

int processFilter(void * work, int size)
{
   struct timeval tbegin,tend;
   char instanceTime[1024];
   int iteracao=0;
   int nu_max_iteracao=0;
   nu_max_iteracao = (int)((Work*)work)->nu_max_iteracao;
   printf("Processing filter0\n");
   gettimeofday(&tbegin,NULL);

   dsReadBuffer(inputstream30P, (&iteracao), sizeof(iteracao));
   while (iteracao<nu_max_iteracao)
   {
      dsWriteBuffer(outputstream5P, (&__new_iteration__), sizeof(
          __new_iteration__));
      dsWriteBuffer(outputstream1P, (&__new_iteration__), sizeof(
          __new_iteration__));
      dsWriteBuffer(outputstream4P, (&__new_iteration__), sizeof(
          __new_iteration__));
      dsWriteBuffer(outputstream0P, (&__new_iteration__), sizeof(
          __new_iteration__));
      dsWriteBuffer(outputstream3P, (&__new_iteration__), sizeof(
          __new_iteration__));
      dsWriteBuffer(outputstream2P, (&__new_iteration__), sizeof(
          __new_iteration__));
      dsWriteBuffer(outputstream6P, (&__new_iteration__), sizeof(
          __new_iteration__));
      dsReadBuffer(inputstream30P, (&iteracao), sizeof(iteracao));
   }
   gettimeofday(&tend,NULL);
   strcpy(instanceTime,elapsed_time(tbegin,tend));
   dsWriteBuffer(outputstream36P, &(instanceTime), sizeof( instanceTime )
       );



   return 0;
}

int finalizeFilter( )
{
   printf("stoping filter0\n");
```

```
    return 0;
}
```

B.2.2.2 filter0.h

```
#ifndef FILTER0_H_
#define FILTER0_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER0_H_*/
```

B.2.2.3 filter1.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>
#include "estruturas-kmeans_vector.h"

#include "work.h"
#include "messages.h"
#include "filter1.h"
#include "util.h"
int __new_iteration__;
InputPortHandler inputstream5P;
OutputPortHandler outputstream37P;
int initFilter(void * work, int size)
{
    printf("Inicializing filter1\n");
    inputstream5P=dsGetInputPortByName("inputstream5P");
    outputstream37P=dsGetOutputPortByName("outputstream37P");
    return 0;
}
```

```
int processFilter(void * work, int size)
{
      struct timeval tbegin,tend;
   char instanceTime[1024];

   printf("Processing filter1\n");
   gettimeofday(&tbegin,NULL);
   while ((dsReadBuffer(inputstream5P, ( & __new_iteration__), sizeof(
       __new_iteration__))!=-2))
   {
      printf("\n\n > Centros --- nova rodada ---------------------- ");
   }
   gettimeofday(&tend,NULL);
   strcpy(instanceTime,elapsed_time(tbegin,tend));
   dsWriteBuffer(outputstream37P, &(instanceTime), sizeof( instanceTime )
       );


   return 0;
}

int finalizeFilter( )
{
   printf("stoping filter1\n");
   return 0;
}
```

B.2.2.4 filter1.h

```
#ifndef FILTER1_H_
#define FILTER1_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER1_H_*/
```

B.2.2.5 filter2.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "messages.h"
#include "filter2.h"
#include "util.h"

int __new_iteration__;
InputPortHandler inputstream1P;
InputPortHandler inputstream14P;
InputPortHandler inputstream23P;
InputPortHandler inputstream32P;
InputPortHandler inputstream34P;
OutputPortHandler outputstream38P;

int initFilter(void * work, int size)
{
   printf("Inicializing filter2\n");
   inputstream34P=dsGetInputPortByName("inputstream34P");
   inputstream32P=dsGetInputPortByName("inputstream32P");
   inputstream23P=dsGetInputPortByName("inputstream23P");
   inputstream14P=dsGetInputPortByName("inputstream14P");
   inputstream1P=dsGetInputPortByName("inputstream1P");
   outputstream38P=dsGetOutputPortByName("outputstream38P");

   return 0;
}

int processFilter(void * work, int size)
{
      struct timeval tbegin,tend;
   char instanceTime[1024];

   printf("Processing filter2\n");

   int   __index_lista_part_centros_x__ = 0 ;
```

```
int *lista_part_centros_x_index ;
int  __index_lista_part_centros_y__ = 0 ;
int *lista_part_centros_y_index ;
float lista_part_centros_x[NUPARTICOES];
     float lista_part_centros_y[NUPARTICOES];
int nu_particoes;
int i;
lista_part_centros_x_index = (int*)malloc(sizeof(int)*NUITENS);
lista_part_centros_y_index = (int*)malloc(sizeof(int)*NUITENS);

message_float _new_message_;

while ( dsReadBuffer(inputstream23P, ( & _new_message_), sizeof(
    _new_message_))!= EOW ) {

   lista_part_centros_x [  __index_lista_part_centros_x__ ] =
       _new_message_.value ;
   lista_part_centros_x_index [  __index_lista_part_centros_x__ ] =
       _new_message_.index ;
   __index_lista_part_centros_x__ ++ ;
}



while ( dsReadBuffer(inputstream14P, ( & _new_message_), sizeof(
    _new_message_))!= EOW ) {
   lista_part_centros_y [  __index_lista_part_centros_y__ ] =
       _new_message_.value ;
   lista_part_centros_y_index [  __index_lista_part_centros_y__ ] =
       _new_message_.index ;
   __index_lista_part_centros_y__ ++ ;
}


gettimeofday(&tbegin,NULL);
nu_particoes = (int)((Work*)work)->nu_particoes;

while ((dsReadBuffer(inputstream1P, ( & __new_iteration__), sizeof(
    __new_iteration__))!=-2))
{
   for (i=0; i<__index_lista_part_centros_x__; i ++ )
   {
      printf("\n -->  (%d) - (%.2f)(%.2f) - Grupo (%d)",
          lista_part_centros_x_index[i], lista_part_centros_x[i],
```

```
            lista_part_centros_y[i], i);

        //LS receive from one
        //index number 0
        dsReadBuffer(inputstream32P, (& _new_message_), sizeof(
            _new_message_)); ;
        lista_part_centros_x [ i ] = _new_message_.value ;
        lista_part_centros_x_index [ i ] = _new_message_.index ;



        //LS receive from one
        //index number 0
        dsReadBuffer(inputstream34P, (& _new_message_), sizeof(
            _new_message_)); ;
        lista_part_centros_y [ i ] = _new_message_.value ;
        lista_part_centros_y_index [ i ] = _new_message_.index ;


      }

   }
   gettimeofday(&tend,NULL);
   strcpy(instanceTime,elapsed_time(tbegin,tend));
   dsWriteBuffer(outputstream38P, &(instanceTime), sizeof( instanceTime )
       );



   return 0;
}

int finalizeFilter( )
{
   printf("stoping filter2\n");
   return 0;
}
```

B.2.2.6 filter2.h

```
#ifndef FILTER2_H_
#define FILTER2_H_
#include "FilterDev.h"
```

```
int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER2_H_*/
```

B.2.2.7 filter3.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "messages.h"
#include "filter3.h"
#include "util.h"

int __new_iteration__;
InputPortHandler inputstream4P;
OutputPortHandler outputstream39P;

int initFilter(void * work, int size)
{
   printf("Inicializing filter3\n");
   inputstream4P=dsGetInputPortByName("inputstream4P");
   outputstream39P=dsGetOutputPortByName("outputstream39P");

   return 0;
}

int processFilter(void * work, int size)
{
      struct timeval tbegin,tend;
   char instanceTime[1024];

   printf("Processing filter3\n");
```

```
    gettimeofday(&tbegin,NULL);

    while ((dsReadBuffer(inputstream4P, ( & __new_iteration__), sizeof(
        __new_iteration__))!=-2))
    {
        // ********************************************************* {}
        // 4 - Atribuir cada amostra a uma cluster de acordo com a  {}
        //       medida de similaridade. {}
        //  {}
        //  Para cada ponto (no centro) : p  { {}
        //    Para cada Centro  : c {   {}
        //      calcula distancia : d = entre c e p {}
        //    }   {}
        //    Atribui p ao cluster C de menor distacia entre c e p {}
        // } {}
        // ********************************************************* {}
        printf("\n -- Calculando os clusters ----------  ");
    }
    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));
    dsWriteBuffer(outputstream39P, &(instanceTime), sizeof( instanceTime )
        );


    return 0;
}

int finalizeFilter(  )
{
    printf("stoping filter3\n");
    return 0;
}
```

B.2.2.8 filter3.h

```
#ifndef FILTER3_H_
#define FILTER3_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);
```

```
int finalizeFilter();

#endif /*FILTER3_H_*/
```

B.2.2.9 filter4.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "work.h"
#include "messages.h"
#include "filter4.h"
#include "util.h"
int __new_iteration__;
InputPortHandler inputstream0P;
InputPortHandler inputstream7P;
InputPortHandler inputstream10P;
InputPortHandler inputstream15P;
InputPortHandler inputstream18P;
InputPortHandler inputstream20P;
InputPortHandler inputstream24P;
OutputPortHandler outputstream27P;
OutputPortHandler outputstream31P;
InputPortHandler inputstream33P;
InputPortHandler inputstream35P;
OutputPortHandler outputstream40P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter4\n");
   inputstream35P=dsGetInputPortByName("inputstream35P");
   inputstream33P=dsGetInputPortByName("inputstream33P");
   outputstream31P=dsGetOutputPortByName("outputstream31P");
   outputstream27P=dsGetOutputPortByName("outputstream27P");
   outputstream40P=dsGetOutputPortByName("outputstream40P");
   inputstream24P=dsGetInputPortByName("inputstream24P");
   inputstream20P=dsGetInputPortByName("inputstream20P");
```

```
    inputstream18P=dsGetInputPortByName("inputstream18P");
    inputstream15P=dsGetInputPortByName("inputstream15P");
    inputstream10P=dsGetInputPortByName("inputstream10P");
    inputstream7P=dsGetInputPortByName("inputstream7P");
    inputstream0P=dsGetInputPortByName("inputstream0P");
    return 0;
}

int processFilter(void * work, int size)
{
       struct timeval tbegin,tend;
    char instanceTime[1024];

    message_float _new_message_float_;
    //message_int  _new_message_int_;
    message_int_array_ptr  _new_message_int_array_;
    int  __index_lista_part_centros_x__ = 0 ;
    int *lista_part_centros_x_index ;
    int  __index_lista_pontos_x__ = 0 ;
    int *lista_pontos_x_index ;
    int  __index_lista_part_centros_y__ = 0 ;
    int *lista_part_centros_y_index ;
    int  __index_lista_pontos_y__ = 0 ;
    int *lista_pontos_y_index ;
    float lista_part_centros_x[NUPARTICOES];
    float lista_part_centros_y[NUPARTICOES];
    float *lista_pontos_x;
    float *lista_pontos_y;
    int nu_pontos;
    int nu_particoes;
    int i,j;
    float x1,x2,y1,y2;
    int menor;
    float lista_distancia[NUPARTICOES];
    int **lista_part_itens;
    int ___i___;

    lista_part_centros_x_index = (int*)malloc(sizeof(int)*NUITENS);
    lista_part_centros_y_index = (int*)malloc(sizeof(int)*NUITENS);
    lista_pontos_x_index = (int*)malloc(sizeof(int)*NUITENS);
    lista_pontos_y_index = (int*)malloc(sizeof(int)*NUITENS);
    lista_pontos_x = (float*)malloc(sizeof(float)*NUITENS);
    lista_pontos_y = (float*)malloc(sizeof(float)*NUITENS);
    lista_part_itens = (int**)malloc(sizeof(int *)*NUPARTICOES);
```

```
_new_message_int_array_=(message_int_array_ptr)malloc(sizeof(
    message_int_array));
for (i=0;i<NUPARTICOES;i++){
    lista_part_itens[i] = (int*)malloc(sizeof(int)*NUITENS);

}




nu_particoes = (int)((Work*)work)->nu_particoes;

printf("Processing filter4\n");



while ( dsReadBuffer(inputstream20P, ( & _new_message_float_), sizeof
    ( _new_message_float_))!= EOW ) {
    lista_pontos_x [ __index_lista_pontos_x__ ] = _new_message_float_
        .value ;
    lista_pontos_x_index [ __index_lista_pontos_x__ ] =
        _new_message_float_.index ;
    __index_lista_pontos_x__ ++ ;
dsReadBuffer(inputstream10P, ( & _new_message_float_), sizeof(
    _new_message_float_));
    lista_pontos_y [ __index_lista_pontos_y__ ] = _new_message_float_
        .value ;
    lista_pontos_y_index [ __index_lista_pontos_y__ ] =
        _new_message_float_.index ;
    __index_lista_pontos_y__ ++ ;
}

while ( dsReadBuffer(inputstream24P, ( & _new_message_float_), sizeof
    ( _new_message_float_))!= EOW ) {
    lista_part_centros_x [ __index_lista_part_centros_x__ ] =
        _new_message_float_.value ;
    lista_part_centros_x_index [ __index_lista_part_centros_x__ ] =
        _new_message_float_.index ;
    __index_lista_part_centros_x__ ++ ;
}
while ( dsReadBuffer(inputstream15P, ( & _new_message_float_), sizeof
    ( _new_message_float_))!= EOW ) {
```

```
        lista_part_centros_y [  __index_lista_part_centros_y__ ] =
            _new_message_float_.value ;
        lista_part_centros_y_index [  __index_lista_part_centros_y__ ] =
            _new_message_float_.index ;
        __index_lista_part_centros_y__ ++ ;
    }


    for(i=0;i<NUPARTICOES;i++){
        dsReadBuffer(inputstream18P, (    _new_message_int_array_), sizeof(
            _new_message_int_array_)) ;
        memcpy(lista_part_itens [ i ] ,  _new_message_int_array_->value,
            sizeof(int)*NUITENS);

    }
// dsReadBuffer(inputstream18P, (& lista_part_itens), sizeof( int )*
    NUPARTICOES*NUITENS);

    /* memcpy(lista_part_itens [  __index_lista_part_itens__ ] ,
        _new_message_int_array_.value,sizeof(int)*NUITENS);
        lista_part_itens_index [  __index_lista_part_itens__ ] =
            _new_message_int_array_.index ;
        dsReadBuffer(inputstream18P, (& _new_message_int_array_), sizeof(
            _new_message_int_array_)) ;
        __index_lista_part_itens__ ++ ;
    }
        __index_lista_part_itens__ ++ ;
*/




        dsReadBuffer(inputstream7P, ( & nu_pontos), sizeof(nu_pontos));
gettimeofday(&tbegin,NULL);

    while ((dsReadBuffer(inputstream0P, ( & __new_iteration__), sizeof(
        __new_iteration__))!=-2))
    {
        // Para cada ponto : i {}



        for (i=0; i<__index_lista_pontos_x__; i ++ )
```

```
{
   x2=lista_pontos_x[i];
   y2=lista_pontos_y[i];
   // Testa se NAo e centro {}
   // A FAZER !!!! {}
   menor=0;
   // Para cada centro: j {}
   for (j=0; j<nu_particoes; j ++ )
   {
      // calcula distancia : d = entre k e i (centro e item) {}

      x1=lista_part_centros_x[j];



      y1=lista_part_centros_y[j];
      lista_distancia[j]=sqrt((((x1-x2)*(x1-x2))+((y1-y2)*(y1-y2)))
          );
      // Garda menor distancia {}
      if (lista_distancia[j]<lista_distancia[menor])
      {
         menor=j;
      }

   }

   // End For j {}
   // Tira ponto de outra particao {}
   for (j=0; j<nu_particoes; j ++ )
   {
      lista_part_itens[j][lista_pontos_x_index[i]]=0;
   }

   // Inclui ponto na partio onde ele tenha menor  {}
   // ditancia ao centro ...  {}
   //

   lista_part_itens[menor][lista_pontos_x_index[i]]=1;



}
   for (j=0; j<nu_particoes; j ++ ){
      _new_message_int_array_->index = j;
```

```
            memcpy(_new_message_int_array_->value , lista_part_itens[j],
                sizeof(int)*NUITENS) ;
        dsWriteBuffer(outputstream31P, (    _new_message_int_array_),
            sizeof( _new_message_int_array_)); ;

        }
        //broadcast receive from all
        for ( ___i___ = 0; ___i___ < dsGetNumWriters( inputstream33P );
            ___i___++) {
            dsReadBuffer(inputstream33P, ( &  _new_message_float_),
                sizeof( _new_message_float_)); ;
            lista_part_centros_x [  ___i___ ] =  _new_message_float_.
                value ;
            lista_part_centros_x_index [  ___i___ ] =
                _new_message_float_.index ;
        }
        //broadcast receive from all
        for ( ___i___ = 0; ___i___ < dsGetNumWriters( inputstream35P );
            ___i___++) {
            dsReadBuffer(inputstream35P, ( &  _new_message_float_),
                sizeof( _new_message_float_)); ;
            lista_part_centros_y [  ___i___ ] =  _new_message_float_.
                value ;
            lista_part_centros_y_index [  ___i___ ] =
                _new_message_float_.index ;
        }

    }
     for(i=0;i<NUPARTICOES;i++){
            memcpy(lista_part_itens [  i ] ,  _new_message_int_array_->
                value,sizeof(int)*NUITENS);
            dsWriteBuffer(outputstream27P, (    _new_message_int_array_),
                sizeof( _new_message_int_array_ ));
        }

    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));
    dsWriteBuffer(outputstream40P, &(instanceTime), sizeof( instanceTime )
        );



    return 0;
}
```

```
int finalizeFilter( )
{
   printf("stoping filter4\n");
   return 0;
}
```

B.2.2.10 filter4.h

```
#ifndef FILTER4_H_
#define FILTER4_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER4_H_*/
```

B.2.2.11 filter5.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "messages.h"
#include "filter5.h"
#include "util.h"
int __new_iteration__;
InputPortHandler inputstream3P;
OutputPortHandler outputstream41P;

int initFilter(void * work, int size)
{
   printf("Inicializing filter5\n");
```

```
    inputstream3P=dsGetInputPortByName("inputstream3P");
    outputstream41P=dsGetOutputPortByName("outputstream41P");

    return 0;
}

int processFilter(void * work, int size)
{
       struct timeval tbegin,tend;
    char instanceTime[1024];

    printf("Processing filter5\n");
       gettimeofday(&tbegin,NULL);

    while ((dsReadBuffer(inputstream3P, ( & __new_iteration__), sizeof(
        __new_iteration__))!=-2))
    {
       // End For i {}
       // ****************************************** {}
       //  5 - Calcular a centride j dos novos clusters  {}
       //  j  a mdia do cluster j, para j=1..K). {}
       //  {}
       //     - Calcula Ponto mdio de cada particao - M {}
       //     - Vefifica se  o final  {}
       //          Se realizou o numero de iteracaoes {}
       // ****************************************** {}
       printf("\n -- Calculando os centroides ----------  ");
    }
gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));
    dsWriteBuffer(outputstream41P, &(instanceTime), sizeof( instanceTime )
        );


    return 0;
}

int finalizeFilter(  )
{
    printf("stoping filter5\n");
    return 0;
}
```

B.2.2.12 filter5.h

```
#ifndef FILTER5_H_
#define FILTER5_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER5_H_*/
```

B.2.2.13 filter6.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "messages.h"
#include "filter6.h"
#include "util.h"

int __new_iteration__;
InputPortHandler inputstream2P;
InputPortHandler inputstream8P;
InputPortHandler inputstream11P;
InputPortHandler inputstream16P;
InputPortHandler inputstream21P;
InputPortHandler inputstream25P;
OutputPortHandler outputstream28P;
OutputPortHandler outputstream29P;
InputPortHandler inputstream31P;
OutputPortHandler outputstream32P;
OutputPortHandler outputstream33P;
OutputPortHandler outputstream34P;
OutputPortHandler outputstream35P;
```

```
OutputPortHandler outputstream42P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter6\n");
   outputstream35P=dsGetOutputPortByName("outputstream35P");
   outputstream34P=dsGetOutputPortByName("outputstream34P");
   outputstream33P=dsGetOutputPortByName("outputstream33P");
   outputstream32P=dsGetOutputPortByName("outputstream32P");
   inputstream31P=dsGetInputPortByName("inputstream31P");
   outputstream29P=dsGetOutputPortByName("outputstream29P");
   outputstream28P=dsGetOutputPortByName("outputstream28P");
   outputstream42P=dsGetOutputPortByName("outputstream42P");
   inputstream25P=dsGetInputPortByName("inputstream25P");
   inputstream21P=dsGetInputPortByName("inputstream21P");
   inputstream16P=dsGetInputPortByName("inputstream16P");
   inputstream11P=dsGetInputPortByName("inputstream11P");
   inputstream8P=dsGetInputPortByName("inputstream8P");
   inputstream2P=dsGetInputPortByName("inputstream2P");
   return 0;
}

int processFilter(void * work, int size)
{
      struct timeval tbegin,tend;
   char instanceTime[1024];

   message_float _new_message_float_;

   float lista_part_centros_x[NUPARTICOES];
   float lista_part_centros_y[NUPARTICOES];
   float *lista_pontos_x;
   float *lista_pontos_y;
   int **lista_part_itens;
   int  __index_lista_part_itens__ = 0 ;
   int *lista_part_itens_index ;
   float soma_x,soma_y,media_x,media_y;
   int itens;



   int nu_particoes;
   int nu_pontos;
   int i,j;
```

```
   int ___i___;
      message_int_array_ptr  _new_message_int_array_;
_new_message_int_array_=(message_int_array_ptr)malloc(sizeof(
    message_int_array));

   printf("Processing filter6\n");
lista_part_itens_index = (int*)malloc(sizeof(int)*NUITENS);

   lista_part_itens = (int**)malloc(sizeof(int *)*NUPARTICOES);
   for (i=0;i<NUPARTICOES;i++){
      lista_part_itens[i] = (int*)malloc(sizeof(int)*NUITENS);

   }
   lista_pontos_x = (float*)malloc(sizeof(float)*NUITENS);
   lista_pontos_y = (float*)malloc(sizeof(float)*NUITENS);

   int  __index_lista_part_centros_x__ = 0 ;
   int *lista_part_centros_x_index ;
   int  __index_lista_pontos_x__ = 0 ;
   int *lista_pontos_x_index;
   int  __index_lista_part_centros_y__ = 0 ;
   int *lista_part_centros_y_index ;
   int  __index_lista_pontos_y__ = 0 ;
   int *lista_pontos_y_index;

   lista_part_centros_x_index = (int*)malloc(sizeof(int)*NUITENS);
   lista_part_centros_y_index = (int*)malloc(sizeof(int)*NUITENS);
   lista_pontos_x_index = (int*)malloc(sizeof(int)*NUITENS);
   lista_pontos_y_index = (int*)malloc(sizeof(int)*NUITENS);



   while ( dsReadBuffer(inputstream21P, ( &  _new_message_float_), sizeof
      ( _new_message_float_))!= EOW ) {
      lista_pontos_x [  __index_lista_pontos_x__ ] =  _new_message_float_
         .value ;
      lista_pontos_x_index [  __index_lista_pontos_x__ ] =
         _new_message_float_.index ;
      __index_lista_pontos_x__ ++ ;
    dsReadBuffer(inputstream11P, ( &  _new_message_float_), sizeof(
       _new_message_float_));
      lista_pontos_y [  __index_lista_pontos_y__ ] =  _new_message_float_
         .value ;
```

```
        lista_pontos_y_index [  __index_lista_pontos_y__ ] =
            _new_message_float_.index ;
        __index_lista_pontos_y__ ++ ;
    }


while ( dsReadBuffer(inputstream25P, ( &  _new_message_float_), sizeof(
    _new_message_float_))!= EOW ) {
        lista_part_centros_x [  __index_lista_part_centros_x__ ] =
            _new_message_float_.value ;
        lista_part_centros_x_index [  __index_lista_part_centros_x__ ] =
            _new_message_float_.index ;
        __index_lista_part_centros_x__ ++ ;
    }



    while ( dsReadBuffer(inputstream16P, ( &  _new_message_float_), sizeof
        ( _new_message_float_))!= EOW ) {
        lista_part_centros_y [  __index_lista_part_centros_y__ ] =
            _new_message_float_.value ;
        lista_part_centros_y_index [  __index_lista_part_centros_y__ ] =
            _new_message_float_.index ;
        __index_lista_part_centros_y__ ++ ;
    }



    nu_particoes = (int)((Work*)work)->nu_particoes;
    dsReadBuffer(inputstream8P, ( & nu_pontos), sizeof(nu_pontos));
    gettimeofday(&tbegin,NULL);

    while ((dsReadBuffer(inputstream2P, ( & __new_iteration__), sizeof(
        __new_iteration__))!=-2))
    {
        // Para cada particao: i {}
        //LS receive from ALL
        //index number 0

        for ( ___i___ = 0; ___i___ < __index_lista_part_centros_x__;___i___
            ++) {
            dsReadBuffer(inputstream31P, (   _new_message_int_array_),
                sizeof( _new_message_int_array_)); ;
            __index_lista_part_itens__ =_new_message_int_array_->index ;
```

```
        memcpy(lista_part_itens [  __index_lista_part_itens__ ] ,
            _new_message_int_array_->value, sizeof(int)*NUITENS);
        lista_part_itens_index [  __index_lista_part_itens__ ] =
            _new_message_int_array_->index ;
    }

    for (i=0; i<__index_lista_part_centros_x__; i ++ )
    {
            soma_x=0;
        soma_y=0;
        itens=0;
        // Para os itens desta partio          {}
        for (j=0; j<nu_pontos; j ++ )
        {

            if (lista_part_itens[i][j]==1)
            {
                soma_x=(soma_x+lista_pontos_x[j]);
                soma_y=(soma_y+lista_pontos_y[j]);
                itens=(itens+1);
            }

        }

        // Este  o ponto mdio do cluster !!      {}
        media_y=(soma_y/itens);
        media_x=(soma_x/itens);
        // Novo centro  o menor              {}
        lista_part_centros_x[i]=media_x;
        lista_part_centros_y[i]=media_y;

        _new_message_float_.index = lista_part_centros_y_index[i] ;
        _new_message_float_.value = lista_part_centros_y[i] ;
        dsWriteBuffer(outputstream35P, ( &  _new_message_float_), sizeof
            ( _new_message_float_)); ;


        _new_message_float_.index = lista_part_centros_y_index[i] ;
        _new_message_float_.value = lista_part_centros_y[i] ;
        dsWriteBuffer(outputstream34P, ( &  _new_message_float_), sizeof
            ( _new_message_float_)); ;

        _new_message_float_.index = lista_part_centros_x_index[i] ;
        _new_message_float_.value = lista_part_centros_x[i] ;
```

```
            dsWriteBuffer(outputstream33P, ( &  _new_message_float_), sizeof
                ( _new_message_float_)); ;

            _new_message_float_.index = lista_part_centros_x_index[i] ;
            _new_message_float_.value = lista_part_centros_x[i] ;
            dsWriteBuffer(outputstream32P, ( &  _new_message_float_), sizeof
                ( _new_message_float_)); ;


        }

    }
        dsWriteBuffer(outputstream28P, ( & lista_part_centros_x), sizeof(
            lista_part_centros_x));
        dsWriteBuffer(outputstream29P, ( & lista_part_centros_y), sizeof(
            lista_part_centros_y));
gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));
    dsWriteBuffer(outputstream42P, &(instanceTime), sizeof( instanceTime )
        );

    return 0;
}

int finalizeFilter( )
{
    printf("stoping filter6\n");
    return 0;
}
```

B.2.2.14 filter6.h

```
#ifndef FILTER6_H_
#define FILTER6_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER6_H_*/
```

B.2.2.15 filter7.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "messages.h"
#include "filter7.h"
#include "util.h"

int __new_iteration__;
InputPortHandler inputstream6P;
OutputPortHandler outputstream30P;
OutputPortHandler outputstream43P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter7\n");
   outputstream30P=dsGetOutputPortByName("outputstream30P");
   outputstream43P=dsGetOutputPortByName("outputstream43P");
   inputstream6P=dsGetInputPortByName("inputstream6P");
   return 0;
}

int processFilter(void * work, int size)
{
      struct timeval tbegin,tend;
   char instanceTime[1024];

   int iteracao=0;
   printf("Processing filter7\n");
   gettimeofday(&tbegin,NULL);
   dsWriteBuffer(outputstream30P, (&iteracao), sizeof(iteracao));

   while ((dsReadBuffer(inputstream6P, (&__new_iteration__), sizeof(
       __new_iteration__))!=-2))
   {
      // End For i {}
      iteracao=(iteracao+1);
      dsWriteBuffer(outputstream30P, (&iteracao), sizeof(iteracao));
```

```
    }
    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));
    dsWriteBuffer(outputstream43P, &(instanceTime), sizeof( instanceTime )
        );



    return 0;
}

int finalizeFilter(  )
{
    printf("stoping filter7\n");
    return 0;
}
```

B.2.2.16 filter7.h

```
#ifndef FILTER7_H_
#define FILTER7_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER7_H_*/
```

B.2.2.17 filter8.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
```

```
#include "messages.h"
#include "filter8.h"
#include "util.h"
int __new_iteration__;
OutputPortHandler outputstream7P;
OutputPortHandler outputstream8P;
OutputPortHandler outputstream9P;
OutputPortHandler outputstream10P;
OutputPortHandler outputstream11P;
OutputPortHandler outputstream12P;
OutputPortHandler outputstream13P;
OutputPortHandler outputstream14P;
OutputPortHandler outputstream15P;
OutputPortHandler outputstream16P;
OutputPortHandler outputstream17P;
OutputPortHandler outputstream18P;
OutputPortHandler outputstream19P;
OutputPortHandler outputstream20P;
OutputPortHandler outputstream21P;
OutputPortHandler outputstream22P;
OutputPortHandler outputstream23P;
OutputPortHandler outputstream24P;
OutputPortHandler outputstream25P;
OutputPortHandler outputstream26P;
OutputPortHandler outputstream44P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter8\n");
   outputstream26P=dsGetOutputPortByName("outputstream26P");
   outputstream25P=dsGetOutputPortByName("outputstream25P");
   outputstream24P=dsGetOutputPortByName("outputstream24P");
   outputstream23P=dsGetOutputPortByName("outputstream23P");
   outputstream22P=dsGetOutputPortByName("outputstream22P");
   outputstream21P=dsGetOutputPortByName("outputstream21P");
   outputstream20P=dsGetOutputPortByName("outputstream20P");
   outputstream19P=dsGetOutputPortByName("outputstream19P");
   outputstream18P=dsGetOutputPortByName("outputstream18P");
   outputstream17P=dsGetOutputPortByName("outputstream17P");
   outputstream16P=dsGetOutputPortByName("outputstream16P");
   outputstream15P=dsGetOutputPortByName("outputstream15P");
   outputstream14P=dsGetOutputPortByName("outputstream14P");
   outputstream13P=dsGetOutputPortByName("outputstream13P");
   outputstream12P=dsGetOutputPortByName("outputstream12P");
   outputstream11P=dsGetOutputPortByName("outputstream11P");
```

```
    outputstream10P=dsGetOutputPortByName("outputstream10P");
    outputstream9P=dsGetOutputPortByName("outputstream9P");
    outputstream8P=dsGetOutputPortByName("outputstream8P");
    outputstream7P=dsGetOutputPortByName("outputstream7P");
    outputstream44P=dsGetOutputPortByName("outputstream44P");
    return 0;
}

int processFilter(void * work, int size)
{
    struct timeval tbegin,tend;
    struct timeval tread;
    char instanceTime[1024];
    int nu_particoes;
    FILE * Arq;
    int nu_pontos;
    char linha[50];
    char str_tmp[50];
    int * lista_pontos_ativo;
    int *lista_pontos_id_item;
    char **lista_pontos_label;
    float *lista_pontos_x;
    float *lista_pontos_y;
    float lista_part_centros_x[NUPARTICOES];
    float lista_part_centros_y[NUPARTICOES];
    int **lista_part_itens;

    message_char  _new_message_char_;
    message_float  _new_message_float_;
       message_int_array_ptr  _new_message_int_array_;
_new_message_int_array_=(message_int_array_ptr)malloc(sizeof(
    message_int_array));


    int m,l,k,t,i,j,ini;
    float x,y;
    char nome_arquivo[1000];
    printf("Processing filter8\n");
    strcpy(nome_arquivo , ((Work*)work)->nome_arquivo);
    gettimeofday(&tbegin,NULL);
lista_pontos_ativo = (int*)malloc(sizeof(int)*NUITENS);
    lista_pontos_x = (float*)malloc(sizeof(float)*NUITENS);
    lista_pontos_y = (float*)malloc(sizeof(float)*NUITENS);
    lista_pontos_id_item = (int*)malloc(sizeof(int)*NUITENS);
```

```
lista_part_itens = (int**)malloc(sizeof(int *)*NUPARTICOES);
for (i=0;i<NUPARTICOES;i++){
    lista_part_itens[i] = (int*)malloc(sizeof(int)*NUITENS);

}
lista_pontos_label = (char**)malloc(sizeof(char *)*NUITENS);
for (i=0;i<NUITENS;i++){
    lista_pontos_label[i] = (char*)malloc(sizeof(char)*50);

}


nu_particoes = (int)((Work*)work)->nu_particoes;

if (Arq=fopen(nome_arquivo, "r+"))
{
    printf("\n --- Ler arquivo - pontos ---");
    nu_pontos=0;
    // Lendo as TUPLAS {}
    for (m=0; ! feof(Arq); m++ )
    {
        linha[0]='\0';
        fgets(linha, 40, Arq);
        l=strlen(linha);
        l--;
        linha[l]='\0';
        // Se linha tem conteudo {}
        if (l>0)
        {
            // Lendo dados da Linha {}
            ini=0;
            for (i=0; i<(2+1); i++)
            {
                bzero(str_tmp, 10);
                // Identifica Item {}
                for (j=ini; j<l; j++)
                {
                    if ((linha[j]==' ')||(j==(l-1)))
                    {
                        k=(j-ini);
                        strncpy(str_tmp, (&linha[ini]), k);
                        ini=(j+1);
                        // Para sair do loop {}
                        j=(l+10);
```

```
    }

}

// Ajuste para atoi {}
t=strlen(str_tmp);
str_tmp[t]='\0';
// Pega Label {}
if (i==0)
{
    lista_pontos_ativo[nu_pontos]=1;
    lista_pontos_id_item[nu_pontos]=nu_pontos;
    strcpy(lista_pontos_label[nu_pontos], str_tmp);

    _new_message_char_.index = nu_pontos ;
    strcpy(_new_message_char_.value, lista_pontos_label[
        nu_pontos] );
    dsWriteBuffer(outputstream13P, ( &  _new_message_char_)
        , sizeof( _new_message_char_)); ;


    // Pega X {}
}
else
{
    if (i==1)
    {
        x=atoi(str_tmp);
        lista_pontos_x[nu_pontos]=x;
            _new_message_float_.index = nu_pontos ;
            _new_message_float_.value = lista_pontos_x[
                nu_pontos] ;
            dsWriteBuffer(outputstream22P, ( &
                _new_message_float_), sizeof(
                _new_message_float_)); ;

            _new_message_float_.index = nu_pontos ;
            _new_message_float_.value = lista_pontos_x[
                nu_pontos] ;
            dsWriteBuffer(outputstream21P, ( &
                _new_message_float_), sizeof(
                _new_message_float_)); ;

            _new_message_float_.index = nu_pontos ;
```

```
                    _new_message_float_.value = lista_pontos_x[
                        nu_pontos] ;
                    dsWriteBuffer(outputstream20P, ( &
                        _new_message_float_), sizeof(
                        _new_message_float_)); ;
                // Pega Y {}
            }
            else
            {
                if (i==2)
                {
                    y=atoi(str_tmp);
                    lista_pontos_y[nu_pontos]=y;
                        _new_message_float_.index = nu_pontos ;
                        _new_message_float_.value = lista_pontos_y[
                            nu_pontos] ;
                        dsWriteBuffer(outputstream12P, ( &
                            _new_message_float_), sizeof(
                            _new_message_float_)); ;

                        _new_message_float_.index = nu_pontos ;
                        _new_message_float_.value = lista_pontos_y[
                            nu_pontos] ;
                        dsWriteBuffer(outputstream11P, ( &
                            _new_message_float_), sizeof(
                            _new_message_float_)); ;

                        _new_message_float_.index = nu_pontos ;
                        _new_message_float_.value = lista_pontos_y[
                            nu_pontos] ;
                        dsWriteBuffer(outputstream10P, ( &
                            _new_message_float_), sizeof(
                            _new_message_float_)); ;
                    // Pega Z {}
                }

            }

        }

    }

    // End For i {}
    // Incrementa {}
```

```
        nu_pontos=(nu_pontos+1);
    }

}
dsWriteBuffer(outputstream9P, ( & nu_pontos), sizeof(nu_pontos));
dsWriteBuffer(outputstream8P, ( & nu_pontos), sizeof(nu_pontos));
dsWriteBuffer(outputstream7P, ( & nu_pontos), sizeof(nu_pontos));
gettimeofday(&tread,NULL);

// End While     (trocado para for) {}
fclose(Arq);
dsCloseOutputPort ( outputstream20P ) ;

dsCloseOutputPort ( outputstream21P ) ;

dsCloseOutputPort ( outputstream22P ) ;
dsCloseOutputPort ( outputstream10P ) ;

dsCloseOutputPort ( outputstream11P ) ;

dsCloseOutputPort ( outputstream12P ) ;


for (i=0; i<nu_pontos; i ++ )
{
    printf("\n --> Ponto:  (%s) - (%.2f)(%.2f) ", lista_pontos_label
        [i], lista_pontos_x[i], lista_pontos_y[i]);
    for (j=0; j<nu_particoes; j ++ ){
        lista_part_itens[j][i]=0;
    }
    lista_part_centros_x[i]=0;
    lista_part_centros_y[i]=0;
}

for (i=0; i<nu_particoes; i ++ )
{
    //choose random point {}
    j=i;
    lista_part_centros_x[i]=lista_pontos_x[j];
    _new_message_float_.index = i ;
    _new_message_float_.value = lista_part_centros_x[i] ;
    dsWriteBuffer(outputstream26P, ( &  _new_message_float_), sizeof
        ( _new_message_float_)); ;
```

```
        dsWriteBuffer(outputstream25P, ( &  _new_message_float_), sizeof
            ( _new_message_float_)); ;
        dsWriteBuffer(outputstream24P, ( &  _new_message_float_), sizeof
            ( _new_message_float_)); ;
        dsWriteBuffer(outputstream23P, ( &  _new_message_float_), sizeof
            ( _new_message_float_)); ;

        lista_part_centros_y[i]=lista_pontos_y[j];
        _new_message_float_.index = i ;
        _new_message_float_.value = lista_part_centros_y[i] ;
        dsWriteBuffer(outputstream17P, ( &  _new_message_float_), sizeof
            ( _new_message_float_)); ;

        dsWriteBuffer(outputstream16P, ( &  _new_message_float_), sizeof
            ( _new_message_float_)); ;

        dsWriteBuffer(outputstream15P, ( &  _new_message_float_), sizeof
            ( _new_message_float_)); ;

        dsWriteBuffer(outputstream14P, ( &  _new_message_float_), sizeof
            ( _new_message_float_)); ;

        lista_part_itens[i][j]=1;
        _new_message_int_array_->index = i ;
        }
dsCloseOutputPort ( outputstream13P ) ;

   dsCloseOutputPort ( outputstream14P ) ;

   dsCloseOutputPort ( outputstream15P ) ;

   dsCloseOutputPort ( outputstream16P ) ;

   dsCloseOutputPort ( outputstream17P ) ;



   dsCloseOutputPort ( outputstream23P ) ;

   dsCloseOutputPort ( outputstream24P ) ;

   dsCloseOutputPort ( outputstream25P ) ;

   dsCloseOutputPort ( outputstream26P ) ;
```

```
      for (i=0; i<nu_particoes; i ++ )
      {
         memcpy(lista_part_itens [ i ] , _new_message_int_array_->value
             ,sizeof(int)*NUITENS);
         dsWriteBuffer(outputstream19P, ( (_new_message_int_array_)),
             sizeof( _new_message_int_array_)); ;

      }
   dsCloseOutputPort ( outputstream19P ) ;

//     _new_message_int_array_->index = j ;
//        memcpy(lista_part_itens [ i ] , _new_message_int_array_.value,
    sizeof(int)*NUITENS);
         //_new_message_int_array_.value = lista_part_itens[i][j] ;
         for(i=0;i<NUPARTICOES;i++){
            memcpy(lista_part_itens [ i ] , _new_message_int_array_->
                value,sizeof(int)*NUITENS);
            dsWriteBuffer(outputstream18P, ( _new_message_int_array_),
                sizeof( _new_message_int_array_ ));
         }

   dsCloseOutputPort ( outputstream18P ) ;




   }

   gettimeofday(&tend,NULL);
   strcpy(instanceTime,elapsed_time(tbegin,tread));
   dsWriteBuffer(outputstream44P, &(instanceTime), sizeof( instanceTime )
       );
   strcpy(instanceTime,elapsed_time(tbegin,tend));
   dsWriteBuffer(outputstream44P, &(instanceTime), sizeof( instanceTime )
       );




      return 0;
}
```

```
int finalizeFilter( )
{
   printf("stoping filter8\n");
   return 0;
}
```

B.2.2.18 filter8.h

```
#ifndef FILTER8_H_
#define FILTER8_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER8_H_*/
```

B.2.2.19 filter9.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "messages.h"
#include "filter9.h"
#include "util.h"
int __new_iteration__;
InputPortHandler inputstream9P;
InputPortHandler inputstream12P;
InputPortHandler inputstream13P;
InputPortHandler inputstream17P;
InputPortHandler inputstream19P;
InputPortHandler inputstream22P;
InputPortHandler inputstream26P;
```

```
InputPortHandler inputstream27P;
InputPortHandler inputstream28P;
InputPortHandler inputstream29P;
InputPortHandler inputstream36P;
InputPortHandler inputstream37P;
InputPortHandler inputstream38P;
InputPortHandler inputstream39P;
InputPortHandler inputstream40P;
InputPortHandler inputstream41P;
InputPortHandler inputstream42P;
InputPortHandler inputstream43P;
InputPortHandler inputstream44P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter9\n");
   inputstream29P=dsGetInputPortByName("inputstream29P");
   inputstream28P=dsGetInputPortByName("inputstream28P");
   inputstream27P=dsGetInputPortByName("inputstream27P");
   inputstream26P=dsGetInputPortByName("inputstream26P");
   inputstream22P=dsGetInputPortByName("inputstream22P");
   inputstream19P=dsGetInputPortByName("inputstream19P");
   inputstream17P=dsGetInputPortByName("inputstream17P");
   inputstream13P=dsGetInputPortByName("inputstream13P");
   inputstream12P=dsGetInputPortByName("inputstream12P");
   inputstream9P=dsGetInputPortByName("inputstream9P");
   inputstream36P=dsGetInputPortByName("inputstream36P");
   inputstream37P=dsGetInputPortByName("inputstream37P");
   inputstream38P=dsGetInputPortByName("inputstream38P");
   inputstream39P=dsGetInputPortByName("inputstream39P");
   inputstream40P=dsGetInputPortByName("inputstream40P");
   inputstream41P=dsGetInputPortByName("inputstream41P");
   inputstream42P=dsGetInputPortByName("inputstream42P");
   inputstream43P=dsGetInputPortByName("inputstream43P");
   inputstream44P=dsGetInputPortByName("inputstream44P");
   return 0;
}

int processFilter(void * work, int size)
{
   struct timeval tbegin, tend;
   char instanceTime[1024];
   char outputFileName[256];
   FILE * Arq;
   InputPortHandler *timeStreams[9];
```

```
    float lista_part_centros_x[NUPARTICOES];
    float lista_part_centros_y[NUPARTICOES];

    int **lista_part_itens;

    char **lista_pontos_label;
    float *lista_pontos_x;
    float *lista_pontos_y;
    int nu_particoes;
    int nu_pontos;
    int i,j;

    int  __index_lista_part_centros_x__ = 0 ;
    int *lista_part_centros_x_index ;

    int  __index_lista_pontos_x__ = 0 ;
    int *lista_pontos_x_index ;
    int  __index_lista_part_itens__ = 0 ;
    int *lista_part_itens_index ;
    int  __index_lista_part_centros_y__ = 0 ;
    int *lista_part_centros_y_index ;
    int  __index_lista_pontos_label__ = 0 ;
    int *lista_pontos_label_index ;
    int  __index_lista_pontos_y__ = 0 ;
    int *lista_pontos_y_index ;
message_float _new_message_float_;
    message_char  _new_message_char_;
    message_int_array_ptr  _new_message_int_array_;
_new_message_int_array_=(message_int_array_ptr)malloc(sizeof(
    message_int_array));


lista_pontos_x = (float*)malloc(sizeof(float)*NUITENS);
    lista_pontos_y = (float*)malloc(sizeof(float)*NUITENS);
    lista_part_itens = (int**)malloc(sizeof(int *)*NUPARTICOES);
    for (i=0;i<NUPARTICOES;i++){
        lista_part_itens[i] = (int*)malloc(sizeof(int)*NUITENS);

    }
    lista_pontos_label = (char**)malloc(sizeof(char *)*NUITENS);
    for (i=0;i<NUITENS;i++){
        lista_pontos_label[i] = (char*)malloc(sizeof(char)*50);

    }
```

```
    lista_part_centros_x_index = (int*)malloc(sizeof(int)*NUITENS);
    lista_part_centros_y_index = (int*)malloc(sizeof(int)*NUITENS);
    lista_pontos_x_index = (int*)malloc(sizeof(int)*NUITENS);
    lista_pontos_y_index = (int*)malloc(sizeof(int)*NUITENS);
    lista_pontos_label_index = (int*)malloc(sizeof(int)*NUITENS);
    lista_part_itens_index = (int*)malloc(sizeof(int)*NUITENS);

    printf("Processing filter9\n");
    gettimeofday(&tbegin,NULL);

while ( dsReadBuffer(inputstream13P, ( & _new_message_char_), sizeof(
    _new_message_char_))!= EOW ) {
        strcpy(lista_pontos_label [ __index_lista_pontos_label__ ],
            _new_message_char_.value );
        lista_pontos_label_index [ __index_lista_pontos_label__ ] =
            _new_message_char_.index ;
        __index_lista_pontos_label__ ++ ;
        dsReadBuffer(inputstream12P, ( & _new_message_float_), sizeof(
            _new_message_float_));
        lista_pontos_y [ __index_lista_pontos_y__ ] = _new_message_float_
            .value ;
        lista_pontos_y_index [ __index_lista_pontos_y__ ] =
            _new_message_float_.index ;
        __index_lista_pontos_y__ ++ ;
    dsReadBuffer(inputstream22P, ( & _new_message_float_), sizeof(
        _new_message_float_));
        lista_pontos_x [ __index_lista_pontos_x__ ] = _new_message_float_
            .value ;
        lista_pontos_x_index [ __index_lista_pontos_x__ ] =
            _new_message_float_.index ;
        __index_lista_pontos_x__ ++ ;
    }
    while ( dsReadBuffer(inputstream26P, ( & _new_message_float_), sizeof
        ( _new_message_float_))!= EOW ) {
        lista_part_centros_x [ __index_lista_part_centros_x__ ] =
            _new_message_float_.value ;
        lista_part_centros_x_index [ __index_lista_part_centros_x__ ] =
            _new_message_float_.index ;
        __index_lista_part_centros_x__ ++ ;
    }
```

```
    while ( dsReadBuffer(inputstream19P, (    _new_message_int_array_),
        sizeof( _new_message_int_array_))!= EOW ) {
        memcpy(lista_part_itens [  __index_lista_part_itens__ ] ,
            _new_message_int_array_->value, sizeof(int)*NUITENS);
        lista_part_itens_index [  __index_lista_part_itens__ ] =
            _new_message_int_array_->index ;
        __index_lista_part_itens__ ++ ;
    }

    while ( dsReadBuffer(inputstream17P, ( &  _new_message_float_), sizeof
        ( _new_message_float_))!= EOW ) {
        lista_part_centros_y [  __index_lista_part_centros_y__ ] =
            _new_message_float_.value ;
        lista_part_centros_y_index [  __index_lista_part_centros_y__ ] =
            _new_message_float_.index ;
        __index_lista_part_centros_y__ ++ ;
    }



    nu_particoes = (int)((Work*)work)->nu_particoes;
    dsReadBuffer(inputstream9P, ( & nu_pontos), sizeof(nu_pontos));
    dsReadBuffer(inputstream29P, ( & lista_part_centros_y), sizeof(
        lista_part_centros_y));
    dsReadBuffer(inputstream28P, ( & lista_part_centros_x), sizeof(
        lista_part_centros_x));
// dsReadBuffer(inputstream27P, ( & lista_part_itens), sizeof(int)*
    NUPARTICOES*NUITENS);
//
    for(i=0;i<NUPARTICOES;i++){
        dsReadBuffer(inputstream27P, (    _new_message_int_array_), sizeof(
            _new_message_int_array_)) ;
        memcpy(lista_part_itens [ i ] ,  _new_message_int_array_->value,
            sizeof(int)*NUITENS);

    }

    {
        printf("\n Resultado ---------- ");
        // Para cada particao: i {}
        for (i=0; i<nu_particoes; i ++ )
        {
            printf("\n particao (%d) - Centro: (%.2f)(%.2f) ", i,
                lista_part_centros_x[i], lista_part_centros_y[i]);
```

```
    // Para cada item: j      {}
    for (j=0; j<nu_pontos; j ++ )
    {
        if (lista_part_itens[i][j]==1)
        {
            printf("\n --> Ponto:(%s) - (%.2f)(%.2f)",
                lista_pontos_label[j], lista_pontos_x[j],
                lista_pontos_y[j]);
        }

    }

}

sprintf (outputFileName,"%s_%d","outputWriterFilter",(int)time(NULL
    ));
Arq=fopen(outputFileName,"w+");
timeStreams[0]=&inputstream36P;
timeStreams[1]=&inputstream37P;
timeStreams[2]=&inputstream38P;
timeStreams[3]=&inputstream39P;
timeStreams[4]=&inputstream40P;
timeStreams[5]=&inputstream41P;
timeStreams[6]=&inputstream42P;
timeStreams[7]=&inputstream43P;
timeStreams[8]=&inputstream44P;
dsReadBuffer(*timeStreams[8], ( & instanceTime ), sizeof(
    instanceTime));
fprintf(Arq,"Filter %d read TIME: %s\n",8,instanceTime);
for (i=0;i<9;i++){


    for (j=0;j<dsGetNumWriters(*timeStreams[i]);j++){
        dsReadBuffer(*timeStreams[i], ( & instanceTime ), sizeof(
            instanceTime));
        fprintf(Arq,"Filter %d instance %d TIME: %s\n",i,j,
            instanceTime);
    }
}

// ------------------------------------------------      {}
printf("\n\n ---- FIM ---- \n");
gettimeofday(&tend,NULL);
fprintf(Arq,"Filter 9 TIME: %s\n",elapsed_time(tbegin,tend));
```

```
        fclose(Arq);

    }

    return 0;
}

int finalizeFilter( )
{
    printf("stoping filter9\n");
    return 0;
}
```

B.2.2.20 filter9.h

```
#ifndef FILTER9_H_
#define FILTER9_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER9_H_*/
```

B.2.2.21 kmeans_vetor.c

```
#include <stdio.h>



#include <stdlib.h>

#include <sys/time.h>
#include <time.h>
#include <string.h>

#include "estruturas-kmeans_vector.h"

#include <math.h>
```

```
#include "work.h"


#include "void.h"

#include "util.h"

int main(int argc, char * argv[])
{
//#pragma ANTHILL {}
   // Arquivo  {}
   char nome_arquivo[1000];
   int nu_particoes;
    struct timeval tbegin, tend;

   int nu_max_iteracao;
    Work *work = (Work *)malloc(sizeof(Work));

   // ------------------------------------------------------------------
         {}
   // Pega parametros {}
   if (argc!=4)
   {
      printf("Executar:  ./a.out <nome arquivo> <# de particoes><#
         interacoes> \n");
      exit(0);
   }

   // passando argumento para inteiro {}
   sprintf(nome_arquivo, "%s", argv[1]);
   sscanf(argv[2], "%d", (&nu_particoes));
   sscanf(argv[3], "%d", (&nu_max_iteracao));
   // ------------------------------------------------------------------
         {}
   printf("\n --- Incio : Arq (%s) - # Particoes (%d) - Int (%d)--",
      nome_arquivo, nu_particoes, nu_max_iteracao);
   // ------------------ Inicia Variavies ------------------------------ {}
   //
   /*
    nu_pontos=0;
      for (i=0; i<20; i ++)
   {
      lista_part_centros_x[i]=0;
```

```
        lista_part_centros_y[i]=0;
        lista_distancia[i]=0;
        for (j=0; j<NUITENS; j ++ )
        {
            lista_part_itens[i][j]=0;
        }

    }*/

    // ****************************************** {}
    //   Inicializa pontos  = arquivo pontos; {}
    //   Inicializa centros = aleatorio; {}
    // ****************************************** {}
    // ------------------ Abre o arquivo ------------------------------
        {}
    // Primeira linha contem os itens ... {}
//#pragma READ BEGIN {}
    /*
        Statement removed because pragma READ annotation found
        */
// iteracao=0;


    work->nu_particoes=nu_particoes;
    work->nu_max_iteracao=nu_max_iteracao;
    strcpy(work->nome_arquivo,nome_arquivo);
    char confFile[] = "./conf.xml";
    Layout *systemLayout = initDs(confFile, argc, argv);
    gettimeofday(&tbegin,NULL);
    appendWork(systemLayout, (void *)work, sizeof(Work));
    gettimeofday(&tend,NULL);
    printf("Total TIME: %s\n",elapsed_time(tbegin,tend));

    finalizeDs(systemLayout);
    /*
#WHILE LOOP REMOVED - FILTERS INSERTED:  0 1 2 3 4 5 6 7
*/
    // End while (teste) {}
    // ------------------------------------------ {}
    // -----     Resultado Final ---------- {}
//#pragma WRITE BEGIN {}
    /*
        Statement removed because pragma WRITE annotation found
        */
```

```
    return 0;
}
```

B.2.2.22 labelFunc.c

```
int hash(char * label, int image)
{
    int dest;
    int * aux;
    aux=(int *)( & label[0]);
    dest=(( * aux)%image);
    return dest;
}

void getLabel(void * msg, int size, char label[])
{
    int * aux;
    aux=(int *)( & label[0]);
    ( * aux)=( * ((int * )msg));
}
```

B.2.2.23 messages.h

```
#include "estruturas-kmeans_vector.h"
struct message_char_t
{
    int index;
    char value[100];
};

typedef struct message_char_t message_char, * message_char_ptr;
struct message_float_t
{
    int index;
    float value;
};

typedef struct message_float_t message_float, * message_float_ptr;
struct message_int_t
{
    int index;
    int value;
```

```
};

typedef struct message_int_t message_int, * message_int_ptr;
struct message_int_array_t
{
   int index;
   int value [NUITENS];
};

typedef struct message_int_array_t message_int_array, *
    message_int_array_ptr;
```

B.2.2.24 work.h

```
struct _Work
{
   int nu_particoes;
   int nu_max_iteracao;
   char nome_arquivo[1000];
};

typedef struct _Work Work, * Work_ptr;
```

B.2.2.25 conf.xml

```
<config>
   <hostdec>
      <host name="uzi01" mem="2048">
         <resource name="1"/>
         </host>
      <host name="uzi02" mem="2048">
         <resource name="2"/>
         </host>
      <host name="uzi03" mem="2048">
         <resource name="3"/>
         </host>
      <host name="uzi05" mem="2048">
         <resource name="4"/>
         </host>
      <host name="uzi08" mem="2048">
         <resource name="5"/>
         </host>
```

```
<host name="uzi09" mem="2048">
   <resource name="6"/>
   </host>
<host name="uzi10" mem="2048">
   <resource name="7"/>
   </host>
<host name="uzi11" mem="2048">
   <resource name="8"/>
   </host>
<host name="uzi12" mem="2048">
   <resource name="9"/>
   </host>
<host name="uzi14" mem="2048">
   <resource name="10"/>
   </host>
      <host name="uzi17" mem="2048">
   <resource name="11"/>
   </host>
<host name="uzi18" mem="2048">
   <resource name="12"/>
   </host>
<host name="uzi19" mem="2048">
   <resource name="13"/>
   </host>
<host name="uzi20" mem="2048">
   <resource name="14"/>
   </host>
<host name="uzi21" mem="2048">
   <resource name="15"/>
   </host>
<host name="uzi22" mem="2048">
   <resource name="16"/>
   </host>
<host name="uzi23" mem="2048">
   <resource name="17"/>
   </host>
<host name="uzi24" mem="2048">
   <resource name="18"/>
   </host>
<host name="uzi25" mem="2048">
   <resource name="19"/>
   </host>
<host name="uzi28" mem="2048">
   <resource name="20"/>
```

```
        </host>
    <host name="uzi32" mem="2048">
        <resource name="21"/>
        </host>
    <host name="uzi33" mem="2048">
        <resource name="22"/>
        </host>
    <host name="uzi35" mem="2048">
        <resource name="23"/>
        </host>
    <host name="uzi06" mem="2048">
        <resource name="24"/>
        </host>
</hostdec>

<placement>
    <filter name="filter0" libname="filter0.so" instances="1">
    <instance demands="1" numinstances="1" />
    </filter>
    <filter name="filter1" libname="filter1.so" instances="1">
    <instance demands="1" numinstances="1" />
    </filter>
    <filter name="filter3" libname="filter3.so" instances="1">
    <instance demands="1" numinstances="1" />
    </filter>
    <filter name="filter5" libname="filter5.so" instances="1">
    <instance demands="1" numinstances="1" />
    </filter>
    <filter name="filter7" libname="filter7.so" instances="1">
    <instance demands="1" numinstances="1" />
    </filter>
    <filter name="filter8" libname="filter8.so" instances="1">
    <instance demands="2" numinstances="1" />
    </filter>
    <filter name="filter9" libname="filter9.so" instances="1">
    <instance demands="1" numinstances="1" />
    </filter>
<filter name="filter2" libname="filter2.so" instances="1">
    <instance demands="3" numinstances="1" />
</filter>

<filter name="filter4" libname="filter4.so" instances="1">
    <instance demands="4" numinstances="1" />
</filter>
```

```
<filter name="filter6" libname="filter6.so" instances="1">
   <instance demands="5" numinstances="1" />
</filter>

</placement>
<layout>
   <stream>
      <from filter="filter0" port="outputstream6P" policy="broadcast"
         />
      <to filter="filter7" port="inputstream6P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream4P" policy="broadcast"
         />
      <to filter="filter3" port="inputstream4P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream2P" policy="broadcast"
         />
      <to filter="filter6" port="inputstream2P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream5P" policy="broadcast"
         />
      <to filter="filter1" port="inputstream5P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream3P" policy="broadcast"
         />
      <to filter="filter5" port="inputstream3P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream1P" policy="broadcast"
         />
      <to filter="filter2" port="inputstream1P"/>
   </stream>
   <stream>
      <from filter="filter0" port="outputstream0P" policy="broadcast"
         />
      <to filter="filter4" port="inputstream0P"/>
   </stream>
   <stream>
```

```
    <from filter="filter4" port="outputstream31P" policy="
        labeled_stream"  policylib="labelFunc.so" />
    <to filter="filter6" port="inputstream31P"/>
  </stream>
  <stream>
    <from filter="filter4" port="outputstream27P" policy="
        round_robin" />
    <to filter="filter9" port="inputstream27P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream35P" policy="broadcast"
         />
    <to filter="filter4" port="inputstream35P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream29P" policy="
        round_robin" />
    <to filter="filter9" port="inputstream29P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream32P" policy="
        labeled_stream"  policylib="labelFunc.so"  />
    <to filter="filter2" port="inputstream32P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream34P" policy="
        labeled_stream"  policylib="labelFunc.so"  />
    <to filter="filter2" port="inputstream34P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream33P" policy="broadcast"
         />
    <to filter="filter4" port="inputstream33P"/>
  </stream>
  <stream>
    <from filter="filter6" port="outputstream28P" policy="
        round_robin" />
    <to filter="filter9" port="inputstream28P"/>
  </stream>
  <stream>
    <from filter="filter7" port="outputstream30P" policy="broadcast"
         />
    <to filter="filter0" port="inputstream30P"/>
  </stream>
```

```
<stream>
   <from filter="filter8" port="outputstream24P" policy="broadcast"
       />
   <to filter="filter4" port="inputstream24P"/>
</stream>
<stream>
   <from filter="filter8" port="outputstream8P" policy="broadcast"
      />
   <to filter="filter6" port="inputstream8P"/>
</stream>
<stream>
   <from filter="filter8" port="outputstream21P" policy="
      round_robin" />
   <to filter="filter6" port="inputstream21P"/>
</stream>
<stream>
   <from filter="filter8" port="outputstream17P" policy="broadcast"
       />
   <to filter="filter9" port="inputstream17P"/>
</stream>
<stream>
   <from filter="filter8" port="outputstream26P" policy="broadcast"
       />
   <to filter="filter9" port="inputstream26P"/>
</stream>
<stream>
   <from filter="filter8" port="outputstream23P" policy="
      labeled_stream"  policylib="labelFunc.so"  />
   <to filter="filter2" port="inputstream23P"/>
</stream>
<stream>
   <from filter="filter8" port="outputstream14P" policy="
      labeled_stream"  policylib="labelFunc.so"  />
   <to filter="filter2" port="inputstream14P"/>
</stream>
<stream>
   <from filter="filter8" port="outputstream9P" policy="broadcast"
      />
   <to filter="filter9" port="inputstream9P"/>
</stream>
<stream>
   <from filter="filter8" port="outputstream7P" policy="broadcast"
      />
   <to filter="filter4" port="inputstream7P"/>
```

```
    </stream>
    <stream>
       <from filter="filter8" port="outputstream22P" policy="
           round_robin" />
       <to filter="filter9" port="inputstream22P"/>
    </stream>
    <stream>
       <from filter="filter8" port="outputstream18P" policy="broadcast"
            />
       <to filter="filter4" port="inputstream18P"/>
    </stream>
    <stream>
       <from filter="filter8" port="outputstream25P" policy="
           labeled_stream"  policylib="labelFunc.so"  />
       <to filter="filter6" port="inputstream25P"/>
    </stream>
    <stream>
       <from filter="filter8" port="outputstream15P" policy="broadcast"
            />
       <to filter="filter4" port="inputstream15P"/>
    </stream>
    <stream>
       <from filter="filter8" port="outputstream11P" policy="
           round_robin" />
       <to filter="filter6" port="inputstream11P"/>
    </stream>
    <stream>
       <from filter="filter8" port="outputstream12P" policy="
           round_robin" />
       <to filter="filter9" port="inputstream12P"/>
    </stream>
    <stream>
       <from filter="filter8" port="outputstream19P" policy="broadcast"
            />
       <to filter="filter9" port="inputstream19P"/>
    </stream>
    <stream>
       <from filter="filter8" port="outputstream10P" policy="
           round_robin" />
       <to filter="filter4" port="inputstream10P"/>
    </stream>
    <stream>
       <from filter="filter8" port="outputstream20P" policy="
           round_robin" />
```

```
    <to filter="filter4" port="inputstream20P"/>
</stream>
<stream>
    <from filter="filter8" port="outputstream13P" policy="
        round_robin" />
    <to filter="filter9" port="inputstream13P"/>
</stream>
<stream>
    <from filter="filter8" port="outputstream16P" policy="
        labeled_stream"  policylib="labelFunc.so"  />
    <to filter="filter6" port="inputstream16P"/>
</stream>
<stream>
    <from filter="filter0" port="outputstream36P" policy="broadcast"
        />
    <to filter="filter9" port="inputstream36P"/>
</stream>
<stream>
    <from filter="filter1" port="outputstream37P" policy="broadcast"
        />
    <to filter="filter9" port="inputstream37P"/>
</stream>
<stream>
    <from filter="filter2" port="outputstream38P" policy="broadcast"
        />
    <to filter="filter9" port="inputstream38P"/>
</stream>
<stream>
    <from filter="filter3" port="outputstream39P" policy="broadcast"
        />
    <to filter="filter9" port="inputstream39P"/>
</stream>
<stream>
    <from filter="filter4" port="outputstream40P" policy="broadcast"
        />
    <to filter="filter9" port="inputstream40P"/>
</stream>
<stream>
    <from filter="filter5" port="outputstream41P" policy="broadcast"
        />
    <to filter="filter9" port="inputstream41P"/>
</stream>
<stream>
```

```
        <from filter="filter6" port="outputstream42P" policy="broadcast"
            />
        <to filter="filter9" port="inputstream42P"/>
      </stream>
      <stream>
        <from filter="filter7" port="outputstream43P" policy="broadcast"
            />
        <to filter="filter9" port="inputstream43P"/>
      </stream>
      <stream>
        <from filter="filter8" port="outputstream44P" policy="broadcast"
            />
        <to filter="filter9" port="inputstream44P"/>
      </stream>
   </layout>
</config>
```

B.3 Knn

B.3.1 Entrada

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#define ROW_SIZE 200
#define COL_SIZE 10

#define MAXLINESIZE 1024




int main(int argc, char ** argv)
{
   char trainFileName [100];
   char testFileName [100];
   char line[MAXLINESIZE];
   char * token;
   float trainData[ROW_SIZE][COL_SIZE];
   float testData[ROW_SIZE][COL_SIZE];
   int trainClass[ROW_SIZE];
```

```
int tempClass[ROW_SIZE];
float distance[ROW_SIZE];
float finalDistance[ROW_SIZE];
float temp_distance;
float temp1;
int testClass[ROW_SIZE];
int dataIndex[ROW_SIZE];
int t;
int k;
int l;
int i;
int j;
FILE * Arq;
int nu_points;
int nu_test;
int nu_dimentions;
int temp_dimentions;
int iteration;
int aux1;
int auxClass1;

int aux2;
int auxClass2;
int auxClass3;

temp1=0.0;
aux1=0;
auxClass1=0;
aux2=0;
auxClass2=0;
auxClass3=0;
trainFileName[0]=0;
testFileName[0]=0;
trainData[0][0]=0;
testData[0][0]=0;
trainClass[0]=0;
tempClass[0]=0;
distance[0]=0;
finalDistance[0]=0;
testClass[0]=0;
dataIndex[0]=0;
t=0;

line[0]=0;
```

```
    k=0;
    l=0;
    iteration=0;
    i=0;
    j=0;
    temp_distance=0.0;
    Arq=NULL;
    token=NULL;
    nu_points=0;
    nu_test=0;
    nu_dimentions=0;
    temp_dimentions=0;



    if(argc<4){
        printf("Usage: %s <train_file> <test_file> <k>\n",argv[0]);
        exit(0);
    }
    strcpy(trainFileName,argv[1]);
    strcpy(testFileName,argv[2]);
    k=atoi(argv[3]);

#ifdef DEBUG
    printf("Arguments:\n\ttrain: %s\n\ttest: %s\n\tk: %d\n",trainFileName,
        testFileName,k);
#endif
    {
        if ( (Arq = fopen(trainFileName,"r+") ) ){
            nu_points =0;
            nu_dimentions =0;
            temp_dimentions = 0;
            for(i=0;(!feof(Arq));i=0){
                line[0]='\0';
                fgets(line,MAXLINESIZE,Arq);
                l = strlen(line);
                //remove \n char
                if(line[l-1]=='\n')l--;
                line[l]='\0';

#ifdef DEBUG
                printf("Line read (size %d): %s\n",l,line);
#endif
```

```
            if(l>0){
                token=strtok(line," ");
                for(i=0;(token!=NULL);i=0){
                    trainData[nu_points][temp_dimentions]=atof(token);
#ifdef DEBUG
                    printf("token read %s\n",token);
#endif
                    temp_dimentions=temp_dimentions+1;
                    token=strtok(NULL," ");

                }
                nu_dimentions=temp_dimentions-1;
                trainClass[nu_points]=(int)trainData[nu_points][
                    nu_dimentions];
#ifdef DEBUG
                printf("Class %d\n",trainClass[nu_points]);
                printf("Nu Dimensoes %d\n",nu_dimentions);
#endif

                nu_points=nu_points+1;

                temp_dimentions=0;
            }









        }

#ifdef DEBUG
        printf("nu_points %d\n",nu_points);
#endif


      }else{
        printf("Cannot open TrainFile: %s\n",trainFileName);
        exit(0);
      }
      fclose(Arq);
      if ( (Arq = fopen(testFileName,"r+") ) ){
        nu_test=0;
        temp_dimentions=0;
```

```
        for(i=0;(!feof(Arq));i=0){
            line[0]='\0';
            fgets(line,MAXLINESIZE,Arq);
            l = strlen(line);
            //remove \n char
            if(line[l-1]=='\n')l--;
            line[l]='\0';
#ifdef DEBUG
            printf("Line read (size %d): %s\n",l,line);
#endif
            if(l>0){
                token=strtok(line," ");
                for(i=0;(token!=NULL);i=0){
                    testData[nu_test][temp_dimentions]=atof(token);
#ifdef DEBUG
                    printf("token read %s\n",token);
#endif
                    temp_dimentions=temp_dimentions+1;

                    token=strtok(NULL," ");

                }
                if(temp_dimentions!=nu_dimentions){
                    printf("Error, wrong number of dimensions on test file:
                        %s\n",testFileName);
                    exit(0);
                }
                temp_dimentions=0;
                nu_test=nu_test+1;
            }
        }
#ifdef DEBUG
        printf("nu_test %d\n",nu_test);
#endif



    }else{
        printf("Cannot open TestFile: %s\n",testFileName);
        exit(0);
    }
    fclose(Arq);
    for (i=0;i<ROW_SIZE;i++){
        testClass[i]=0;
```

```
            distance[i]=0;
            dataIndex[i]=0;
            tempClass[i]=0;
            finalDistance[i]=0;
        }
    }
    iteration=0;


    while(iteration<nu_test){
        for(i=0;i<nu_points;i++){
            temp_distance=0.0;
            for(j=0;j<nu_dimentions;j++){
                temp_distance+=pow((trainData[i][j]-testData[iteration][j])
                    ,2);
#ifdef DEBUG
                printf("temp distance is %f\n",temp_distance);
#endif
            }
            distance[i]=pow(temp_distance,0.5);
#ifdef DEBUG
                printf("distance is %f\n",distance[i]);
#endif

            dataIndex[i]=i;
        }
        for(i=0;i<nu_points;i++){
            for(j=i+1;j<nu_points;j++){
                if(distance[j]<distance[i]){
                    temp1=distance[i];
                    distance[i]=distance[j];
                    distance[j]=temp1;
                    t=dataIndex[i];
                    dataIndex[i]=dataIndex[j];
                    dataIndex[j]=t;
                }
            }
        }
        {
        for(i=0;i<k;i++){
            aux1=dataIndex[i];
            auxClass1 = trainClass[aux1];
            tempClass[auxClass1]=0;
        }
```

```
        for(i=0;i<k;i++){
#ifdef DEBUG
            printf("Class %d\n",trainClass[dataIndex[i]]);
#endif
            aux2=dataIndex[i];
            auxClass2=trainClass[aux2];
            tempClass[auxClass2]=tempClass[auxClass2]+1;
            auxClass3 = testClass[iteration];
            if(tempClass[auxClass2]>tempClass[auxClass3]){
                testClass[iteration]=auxClass2;
            }


        }
        finalDistance[iteration]=distance[0];
        }
        iteration=iteration+1;
    }

    {
        for(i=0;i<nu_test;i++){
            printf("Distance: %f\tClass: %d\n",finalDistance[i],testClass[i
                ]);

        }
    }


    return 0;
}
```

B.3.2 Saída

B.3.2.1 filter0.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

```
#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter0.h"
int __new_iteration__;
OutputPortHandler outputstream0P;
OutputPortHandler outputstream1P;
OutputPortHandler outputstream2P;
OutputPortHandler outputstream3P;
InputPortHandler inputstream5P;
InputPortHandler inputstream19P;
OutputPortHandler outputstream26P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter0\n");
   outputstream26P=dsGetOutputPortByName("outputstream26P");
   inputstream19P=dsGetInputPortByName("inputstream19P");
   inputstream5P=dsGetInputPortByName("inputstream5P");
   outputstream3P=dsGetOutputPortByName("outputstream3P");
   outputstream2P=dsGetOutputPortByName("outputstream2P");
   outputstream1P=dsGetOutputPortByName("outputstream1P");
   outputstream0P=dsGetOutputPortByName("outputstream0P");
   return 0;
}

int processFilter(void * work, int size)
{
   int iteration;
   int nu_test;
   printf("Processing filter0\n");
   struct timeval tbegin,tend;
   char instanceTime[1024];
   dsReadBuffer(inputstream5P, (&nu_test), sizeof(nu_test));
   dsReadBuffer(inputstream19P, (&iteration), sizeof(iteration));
   gettimeofday(&tbegin,NULL);
   while (iteration<nu_test)
   {
      dsWriteBuffer(outputstream0P, (&__new_iteration__), sizeof(
         __new_iteration__));
      dsWriteBuffer(outputstream1P, (&__new_iteration__), sizeof(
         __new_iteration__));
      dsWriteBuffer(outputstream3P, (&__new_iteration__), sizeof(
         __new_iteration__));
```

```
        dsWriteBuffer(outputstream2P, (&__new_iteration__), sizeof(
            __new_iteration__));
        dsReadBuffer(inputstream19P, (&iteration), sizeof(iteration));
    }

    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));

    dsWriteBuffer(outputstream26P, (&instanceTime), sizeof(instanceTime)
        );
    return 0;
}

int finalizeFilter( )
{
    printf("stoping filter0\n");
    return 0;
}
```

B.3.2.2 filter0.h

```
#ifndef FILTER0_H_
#define FILTER0_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER0_H_*/
```

B.3.2.3 filter1.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>
```

```
#include <math.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter1.h"
int __new_iteration__;
InputPortHandler inputstream0P;
InputPortHandler inputstream7P;
InputPortHandler inputstream9P;
InputPortHandler inputstream10P;
InputPortHandler inputstream11P;
InputPortHandler inputstream12P;
InputPortHandler inputstream13P;
InputPortHandler inputstream17P;
OutputPortHandler outputstream18P;
InputPortHandler inputstream20P;
InputPortHandler inputstream22P;
OutputPortHandler outputstream23P;
OutputPortHandler outputstream27P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter1\n");
   outputstream27P=dsGetOutputPortByName("outputstream27P");
   outputstream23P=dsGetOutputPortByName("outputstream23P");
   inputstream22P=dsGetInputPortByName("inputstream22P");
   inputstream20P=dsGetInputPortByName("inputstream20P");
   outputstream18P=dsGetOutputPortByName("outputstream18P");
   inputstream17P=dsGetInputPortByName("inputstream17P");
   inputstream13P=dsGetInputPortByName("inputstream13P");
   inputstream12P=dsGetInputPortByName("inputstream12P");
   inputstream11P=dsGetInputPortByName("inputstream11P");
   inputstream10P=dsGetInputPortByName("inputstream10P");
   inputstream9P=dsGetInputPortByName("inputstream9P");
   inputstream7P=dsGetInputPortByName("inputstream7P");
   inputstream0P=dsGetInputPortByName("inputstream0P");
   return 0;
}

int processFilter(void * work, int size)
{
   printf("Processing filter1\n");
   struct timeval tbegin,tend;
   char instanceTime[1024];
```

```
float ** trainData;
float **testData;
float * distance;
int * dataIndex;

int i,j,iteration;
int nu_points;
float temp_distance;
int nu_dimentions;

//using variable testData#5
int __index_testData__ = 0 ;
int * testData_index ;
message_float * _new_message_float_;
message_float_array *_new_message_float_array_;
message_int * _new_message_int_;
int __index_distance__ = 0 ;
int * distance_index ;
int __index_trainData__ = 0 ;
int * trainData_index ;
trainData_index =(int*)malloc(sizeof(int)*ROW_SIZE);
int __index_dataIndex__ = 0 ;
int * dataIndex_index ;
trainData = (float **)malloc(sizeof(float *)*ROW_SIZE);
for (i=0;i<ROW_SIZE;i++){
   trainData[i]=(float *)malloc(sizeof(float)*COL_SIZE);
}
testData = (float **)malloc(sizeof(float *)*ROW_SIZE);
for (i=0;i<ROW_SIZE;i++){
   testData[i]=(float *)malloc(sizeof(float)*COL_SIZE);
}
distance = (float *)malloc(sizeof(float )*ROW_SIZE);
dataIndex = (int *)malloc(sizeof(int )*ROW_SIZE);
testData_index =(int*)malloc(sizeof(int)*ROW_SIZE);
dataIndex_index =(int*)malloc(sizeof(int)*ROW_SIZE);
distance_index =(int*)malloc(sizeof(int)*ROW_SIZE);
_new_message_float_ = (message_float *)malloc(sizeof(message_float));
_new_message_float_array_ = (message_float_array *)malloc (sizeof(
    message_float_array ));
 _new_message_int_ = ( message_int*)malloc (sizeof( message_int ));
//using variable trainData#5
while ( dsReadBuffer(inputstream9P, ( _new_message_float_array_),
    sizeof( message_float_array))!= EOW ) {
```

```
    memcpy(trainData [ __index_trainData__ ] ,
        _new_message_float_array_->value ,sizeof(float)*COL_SIZE);
    trainData_index [ __index_trainData__ ] =
        _new_message_float_array_->index ;
    __index_trainData__ ++ ;
}


dsReadBuffer(inputstream10P , ( & nu_dimentions), sizeof(nu_dimentions)
    );
dsReadBuffer(inputstream13P , ( & nu_points), sizeof(nu_points));

while ( dsReadBuffer(inputstream12P , (  _new_message_float_array_),
    sizeof( message_float_array))!= EOW ) {
    memcpy(testData [ __index_testData__ ] , _new_message_float_array_
        ->value ,sizeof(float)*COL_SIZE);
    testData_index [ __index_testData__ ] =  _new_message_float_array_
        ->index ;
    __index_testData__ ++ ;
}

//using variable distance#11
while ( dsReadBuffer(inputstream11P , (  _new_message_float_), sizeof(
     message_float))!= EOW ) {
    distance [ __index_distance__ ] =  _new_message_float_->value ;
    distance_index [ __index_distance__ ] =  _new_message_float_->
        index ;
    __index_distance__ ++ ;
}

//using variable dataIndex#11
while ( dsReadBuffer(inputstream7P , (  _new_message_int_), sizeof(
    message_int))!= EOW ) {
    dataIndex [ __index_dataIndex__ ] =  _new_message_int_->value ;
    dataIndex_index [ __index_dataIndex__ ] =  _new_message_int_->
        index ;
    __index_dataIndex__ ++ ;
}

gettimeofday(&tbegin ,NULL);
while ((dsReadBuffer(inputstream0P , ( & __new_iteration__), sizeof(
    __new_iteration__))!=-2))
{
    /*
```

```
        //LS receive from one
        //index number 0//using variable distance#6
        dsReadBuffer(inputstream22P, (   _new_message_float_), sizeof(
            _new_message_float_)); ;
        distance [  __index_distance__ ] =  _new_message_float_->value ;
        distance_index [  __index_distance__ ] =  _new_message_float_->
            index ;

              //LS receive from one
        //index number 0//using variable dataIndex#6
        dsReadBuffer(inputstream17P, (   _new_message_int_), sizeof(
            _new_message_int_)); ;
        dataIndex [  __index_dataIndex__ ] =  _new_message_int_->value ;
        dataIndex_index [  __index_dataIndex__ ] =  _new_message_int_->
            index ;
*/
        dsReadBuffer(inputstream20P, ( & iteration), sizeof(iteration));
        printf("iteration %d\n",iteration);
        if(iteration == EOW){
           break;
        }

        for (i=0; i<__index_trainData__; i ++ )
        {
           temp_distance=0.0;
           for (j=0; j<nu_dimentions; j ++ )
           {
              temp_distance+=pow((trainData[i][j]-testData[iteration][j]),
                  2);
           }

           distance[i]=pow(temp_distance, 0.5);
           //RHS  trocado de i para trainData_index[i]
           dataIndex[i]=trainData_index[i];
           //RHS  trocado de i para trainData_index[i]
           _new_message_float_->index = trainData_index[i] ;
           _new_message_float_->value = distance[i] ;
           dsWriteBuffer(outputstream23P, (   _new_message_float_), sizeof(
               message_float)); ;
           //RHS  trocado de i para trainData_index[i]
           _new_message_int_->index = trainData_index[i];
           _new_message_int_->value = dataIndex[i] ;
           dsWriteBuffer(outputstream18P, (   _new_message_int_), sizeof(
              message_int)); ;
```

```
        }

    }

    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));

    dsWriteBuffer(outputstream27P, (&instanceTime), sizeof(instanceTime)
        );
    for (i=0;i<ROW_SIZE;i++){
        free(trainData[i]);
    }
    free(trainData);
    for (i=0;i<ROW_SIZE;i++){
        free(testData[i]);
    }
    free( testData );
    free( distance );
    free( dataIndex );
    free( testData_index );
    free( dataIndex_index );
    free( distance_index );
    return 0;
}

int finalizeFilter( )
{
    printf("stoping filter1\n");
    return 0;
}
```

B.3.2.4 filter1.h

```
#ifndef FILTER1_H_
#define FILTER1_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);
```

```
int finalizeFilter();

#endif /*FILTER1_H_*/
```

B.3.2.5 filter2.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter2.h"
int __new_iteration__;
InputPortHandler inputstream1P;
InputPortHandler inputstream14P;
OutputPortHandler outputstream17P;
InputPortHandler inputstream18P;
InputPortHandler inputstream32P;
InputPortHandler inputstream33P;
OutputPortHandler outputstream22P;
InputPortHandler inputstream23P;
OutputPortHandler outputstream24P;
OutputPortHandler outputstream25P;
OutputPortHandler outputstream28P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter2\n");
   outputstream28P=dsGetOutputPortByName("outputstream28P");
   outputstream25P=dsGetOutputPortByName("outputstream25P");
   outputstream24P=dsGetOutputPortByName("outputstream24P");
   inputstream23P=dsGetInputPortByName("inputstream23P");
   inputstream32P=dsGetInputPortByName("inputstream32P");
   inputstream33P=dsGetInputPortByName("inputstream33P");
   outputstream22P=dsGetOutputPortByName("outputstream22P");
   inputstream18P=dsGetInputPortByName("inputstream18P");
   outputstream17P=dsGetOutputPortByName("outputstream17P");
   inputstream14P=dsGetInputPortByName("inputstream14P");
```

```
    inputstream1P=dsGetInputPortByName("inputstream1P");
    return 0;
}

int processFilter(void * work, int size)
{
    printf("Processing filter2\n");
    struct timeval tbegin,tend;
    char instanceTime[1024];
    float *distance;
    int i,j;
    int *dataIndex;
    float temp1;
    int t;
    int k;
    int nu_points;
    message_float * _new_message_float_;
    int __index_distance__ = 0 ;
    int * distance_index ;
    int ___i___ ;
    message_int * _new_message_int_;
    int __index_dataIndex__ = 0 ;
    int * dataIndex_index ;
    dataIndex_index =(int*)malloc(sizeof(int)*ROW_SIZE);
    distance = (float *)malloc(sizeof(float )*ROW_SIZE);
    dataIndex = (int *)malloc(sizeof(int )*ROW_SIZE);
    distance_index =(int*)malloc(sizeof(int)*ROW_SIZE);
  _new_message_float_ = (message_float *)malloc(sizeof(message_float));
          _new_message_int_ = ( message_int*)malloc (sizeof( message_int )
              );

    dsReadBuffer(inputstream14P, ( & nu_points), sizeof(nu_points));

    while ( dsReadBuffer(inputstream32P, ( _new_message_float_), sizeof(
        message_float))!= EOW ) {
       distance [ __index_distance__ ] = _new_message_float_->value ;
       distance_index [ __index_distance__ ] = _new_message_float_->
           index ;
       __index_distance__ ++ ;
       dsReadBuffer(inputstream33P, ( _new_message_int_), sizeof(
           message_int));
       dataIndex [ __index_dataIndex__ ] = _new_message_int_->value ;
       dataIndex_index [ __index_dataIndex__ ] = _new_message_int_->
           index ;
```

```
    __index_dataIndex__ ++ ;

}

k = (int)((Work*)work)->k;


gettimeofday(&tbegin,NULL);
while ((dsReadBuffer(inputstream1P, ( & __new_iteration__), sizeof(
    __new_iteration__))!=-2))
{
    //broadcast receive from all
    //using variable distance#4
    for (___i___ = 0; ___i___ < __index_distance__;___i___++) {
        dsReadBuffer(inputstream23P, (    _new_message_float_), sizeof(
            message_float)); ;
        distance [  ___i___ ] =  _new_message_float_->value ;
        distance_index [  ___i___  ] =  _new_message_float_->index ;
        dsReadBuffer(inputstream18P, (    _new_message_int_), sizeof(
            message_int)); ;
        dataIndex [ ___i___ ] =  _new_message_int_->value ;
        dataIndex_index [  ___i___ ] =  _new_message_int_->index ;
    }

    for (i=0; i<__index_distance__; i ++ )
    {
        for (j=(i+1); j<__index_distance__; j ++ )
        {
            if (distance[j]<distance[i])
            {
                temp1=distance[i];
                distance[i]=distance[j];
                distance[j]=temp1;
                t=dataIndex[i];
                dataIndex[i]=dataIndex[j];
                dataIndex[j]=t;

            }

        }

    }
    dsWriteBuffer(outputstream25P, (  distance), sizeof(float)*k);
    dsWriteBuffer(outputstream24P, (  dataIndex), sizeof(int)*k);
```

```
    }

    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));

    dsWriteBuffer(outputstream28P, (&instanceTime), sizeof(instanceTime)
        );
    free (dataIndex_index  );
    free (distance );
    free (dataIndex );
    free (distance_index  );

    return 0;
}

int finalizeFilter( )
{
    printf("stoping filter2\n");
    return 0;
}
```

B.3.2.6 filter2.h

```
#ifndef FILTER2_H_
#define FILTER2_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER2_H_*/
```

B.3.2.7 filter3.c

```
#include "FilterDev.h"

#include <stdio.h>
```

```
#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter3.h"
int __new_iteration__;
InputPortHandler inputstream3P;
OutputPortHandler outputstream15P;
OutputPortHandler outputstream16P;
InputPortHandler inputstream21P;
InputPortHandler inputstream24P;
InputPortHandler inputstream25P;
InputPortHandler inputstream34P;
OutputPortHandler outputstream29P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter3\n");
   outputstream29P=dsGetOutputPortByName("outputstream29P");
   inputstream25P=dsGetInputPortByName("inputstream25P");
   inputstream24P=dsGetInputPortByName("inputstream24P");
   inputstream34P=dsGetInputPortByName("inputstream34P");
   inputstream21P=dsGetInputPortByName("inputstream21P");
   outputstream16P=dsGetOutputPortByName("outputstream16P");
   outputstream15P=dsGetOutputPortByName("outputstream15P");
   inputstream3P=dsGetInputPortByName("inputstream3P");
   return 0;
}

int processFilter(void * work, int size)
{
   printf("Processing filter3\n");
   struct timeval tbegin,tend;
   char instanceTime[1024];
   int i;
   int k;
   int j,t;
   float temp1;
   int iteration;
   int aux1;
   int auxClass1;
   int ___i___;
```

```
int aux2;
int auxClass2;
int auxClass3;
int *trainClass;
int *tempClass;
int *testClass;
int *dataIndex;
float *distance;
float *finalDistance;
int  __index_trainClass__ = 0 ;
int * trainClass_index ;
trainClass_index  =(int*)malloc(sizeof(int)*ROW_SIZE);
distance = (float *)malloc(sizeof(float )*ROW_SIZE);
finalDistance = (float *)malloc(sizeof(float )*ROW_SIZE);
trainClass = (int *)malloc(sizeof(int )*ROW_SIZE);
tempClass = (int *)malloc(sizeof(int )*ROW_SIZE);
testClass = (int *)malloc(sizeof(int )*ROW_SIZE);
dataIndex = (int *)malloc(sizeof(int )*ROW_SIZE);
int num_writers = dsGetNumWriters(inputstream25P);


message_int *  _new_message_int_;
 _new_message_int_ = ( message_int*)malloc (sizeof( message_int ));

k = (int)((Work*)work)->k;
for (i=0; i<ROW_SIZE; i ++ ) {
   testClass[i]=0;
}

while ( dsReadBuffer(inputstream34P, (    _new_message_int_), sizeof(
    message_int))!= EOW ) {
   trainClass [  __index_trainClass__ ] =  _new_message_int_->value ;
   trainClass_index [  __index_trainClass__ ] =  _new_message_int_->
       index ;
   __index_trainClass__ ++ ;
}
gettimeofday(&tbegin,NULL);

while ((dsReadBuffer(inputstream3P, ( & __new_iteration__), sizeof(
    __new_iteration__))!=-2))
{
   {
      dsReadBuffer(inputstream21P, ( & iteration), sizeof(iteration));
```

```
//correcao, pegando somente k elementos de cada instancia de
    filtro.
for(___i___=0;___i___<num_writers;___i___++){
    dsReadBuffer(inputstream25P, (&distance[___i___ * k]),
        sizeof(float)*k);
    dsReadBuffer(inputstream24P, (&dataIndex[___i___*k]),
        sizeof(int)*k);
}
//retirado do filtro 2, realizando a reducao global da
    informacao
for (i=0; i<num_writers*k; i ++ )
{
    for (j=(i+1); j<num_writers*k; j ++ )
    {
        if (distance[j]<distance[i])
        {
            temp1=distance[i];
            distance[i]=distance[j];
            distance[j]=temp1;
            t=dataIndex[i];
            dataIndex[i]=dataIndex[j];
            dataIndex[j]=t;

        }

    }

}

for (i=0; i<k; i ++ )
{
    aux1=dataIndex[i];
    auxClass1=trainClass[aux1];
    tempClass[auxClass1]=0;
}

for (i=0; i<k; i ++ )
{
    aux2=dataIndex[i];
    auxClass2=trainClass[aux2];
    tempClass[auxClass2]=(tempClass[auxClass2]+1);
    auxClass3=testClass[iteration];
    if (tempClass[auxClass2]>tempClass[auxClass3])
    {
```

```
                testClass[iteration]=auxClass2;
            }

        }

        finalDistance[iteration]=distance[0];
    }

}

dsWriteBuffer(outputstream15P, (  finalDistance), sizeof(float)*
    ROW_SIZE);
dsWriteBuffer(outputstream16P, (  testClass), sizeof(int)*ROW_SIZE);
gettimeofday(&tend,NULL);
strcpy(instanceTime,elapsed_time(tbegin,tend));

dsWriteBuffer(outputstream29P, ( & instanceTime), sizeof(instanceTime)
    );
free( trainClass_index);
free( distance );
free( finalDistance );
free( trainClass );
free( tempClass );
free( testClass );
free( dataIndex );


return 0;
}

int finalizeFilter(  )
{
printf("stoping filter3\n");
return 0;
}
```

B.3.2.8 filter3.h

```
#ifndef FILTER3_H_
#define FILTER3_H_
#include "FilterDev.h"

int initFilter(void *work, int size);
```

```
int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER3_H_*/
```

B.3.2.9 filter4.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter4.h"
int __new_iteration__;
InputPortHandler inputstream2P;
OutputPortHandler outputstream19P;
OutputPortHandler outputstream20P;
OutputPortHandler outputstream21P;
OutputPortHandler outputstream30P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter4\n");
   outputstream30P=dsGetOutputPortByName("outputstream30P");
   outputstream21P=dsGetOutputPortByName("outputstream21P");
   outputstream20P=dsGetOutputPortByName("outputstream20P");
   outputstream19P=dsGetOutputPortByName("outputstream19P");
   inputstream2P=dsGetInputPortByName("inputstream2P");
   return 0;
}

int processFilter(void * work, int size)
{
   printf("Processing filter4\n");
      struct timeval tbegin,tend;
      char instanceTime[1024];
```

```
    int iteration=0;
    gettimeofday(&tbegin,NULL);
    dsWriteBuffer(outputstream19P, (&iteration), sizeof(iteration));
    dsWriteBuffer(outputstream20P, (&iteration), sizeof(iteration));
    dsWriteBuffer(outputstream21P, (&iteration), sizeof(iteration));
    iteration=(iteration+1);
    while ((dsReadBuffer(inputstream2P, (&__new_iteration__), sizeof(
        __new_iteration__))!=-2))
    {
       dsWriteBuffer(outputstream19P, (&iteration), sizeof(iteration)
           );
       dsWriteBuffer(outputstream20P, (&iteration), sizeof(iteration)
           );
       dsWriteBuffer(outputstream21P, (&iteration), sizeof(iteration)
           );
       iteration=(iteration+1);
    }

    gettimeofday(&tend,NULL);
    strcpy(instanceTime,elapsed_time(tbegin,tend));

    dsWriteBuffer(outputstream30P, (&instanceTime), sizeof(
        instanceTime));
    return 0;
}

int finalizeFilter( )
{
   printf("stoping filter4\n");
   return 0;
}
```

B.3.2.10 filter4.h

```
#ifndef FILTER4_H_
#define FILTER4_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();
```

```
#endif /*FILTER4_H_*/
```

B.3.2.11 filter5.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter5.h"
int __new_iteration__;
OutputPortHandler outputstream5P;
OutputPortHandler outputstream6P;
OutputPortHandler outputstream7P;
OutputPortHandler outputstream9P;
OutputPortHandler outputstream10P;
OutputPortHandler outputstream11P;
OutputPortHandler outputstream12P;
OutputPortHandler outputstream13P;
OutputPortHandler outputstream14P;
OutputPortHandler outputstream34P;
OutputPortHandler outputstream31P;
OutputPortHandler outputstream32P;
OutputPortHandler outputstream33P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter5\n");
   outputstream31P=dsGetOutputPortByName("outputstream31P");
   outputstream32P=dsGetOutputPortByName("outputstream32P");
   outputstream33P=dsGetOutputPortByName("outputstream33P");
   outputstream14P=dsGetOutputPortByName("outputstream14P");
   outputstream34P=dsGetOutputPortByName("outputstream34P");
   outputstream13P=dsGetOutputPortByName("outputstream13P");
   outputstream12P=dsGetOutputPortByName("outputstream12P");
   outputstream11P=dsGetOutputPortByName("outputstream11P");
   outputstream10P=dsGetOutputPortByName("outputstream10P");
```

```
    outputstream9P=dsGetOutputPortByName("outputstream9P");
    outputstream7P=dsGetOutputPortByName("outputstream7P");
    outputstream6P=dsGetOutputPortByName("outputstream6P");
    outputstream5P=dsGetOutputPortByName("outputstream5P");
    return 0;
}

int processFilter(void * work, int size)
{
    printf("Processing filter5\n");
    struct timeval tbegin,tend;
    gettimeofday(&tbegin,NULL);

    char instanceTime[1024];

    char trainFileName [1024];
    char testFileName [1024];
    char line[MAXLINESIZE];
    char * token;
    float **trainData;
    float **testData;
    int  *trainClass;
    float *distance;
    int *dataIndex;
    int l;
    int i;
    FILE * Arq;
    int nu_points;
    int nu_test;
    int nu_dimentions;
    int temp_dimentions;
    message_float * _new_message_float_;
    message_float_array * _new_message_float_array_;
    message_int * _new_message_int_;
    trainData = (float **)malloc(sizeof(float *)*ROW_SIZE);
    for (i=0;i<ROW_SIZE;i++){
        trainData[i]=(float *)malloc(sizeof(float)*COL_SIZE);
    }
    testData = (float **)malloc(sizeof(float *)*ROW_SIZE);
    for (i=0;i<ROW_SIZE;i++){
        testData[i]=(float *)malloc(sizeof(float)*COL_SIZE);
    }
    distance = (float *)malloc(sizeof(float )*ROW_SIZE);
    dataIndex = (int *)malloc(sizeof(int )*ROW_SIZE);
```

```
trainClass = (int *)malloc(sizeof(int )*ROW_SIZE);
 _new_message_float_ = (message_float *)malloc(sizeof(message_float));
     _new_message_float_array_ = (message_float_array *)malloc (sizeof
        ( message_float_array ));
      _new_message_int_ = ( message_int*)malloc (sizeof( message_int )
         );




strcpy (testFileName , (char *)((Work*)work)->testFileName);
strcpy (trainFileName , (char *)((Work*)work)->trainFileName);
{
   if (Arq=fopen(trainFileName , "r+"))
   {
      nu_points=0;
      nu_dimentions=0;
      temp_dimentions=0;
      for (i=0; ! feof(Arq); i=0) {
         line[0]='\0';
         fgets(line , 1024, Arq);
         l=strlen(line);
         //remove \n char {}
         if (line[(l-1)]=='\n')
         {
            l -- ;
         }

         line[l]='\0';
         if (l>0)
         {
            token=strtok(line , " ");
            for (i=0; token!=((void * )0); i=0)
            {
               trainData[nu_points][temp_dimentions]=atof(token);
               temp_dimentions=(temp_dimentions+1);
               token=strtok(((void * )0), " ");
            }

            nu_dimentions=(temp_dimentions-1);
            trainClass[nu_points]=((int)trainData[nu_points][
               nu_dimentions]);
            _new_message_float_array_->index = nu_points ;
```

```
            memcpy (_new_message_float_array_->value , trainData[
                nu_points],sizeof(float)*COL_SIZE) ;
            dsWriteBuffer(outputstream9P, (
                _new_message_float_array_), sizeof( message_float_array
                ));


            _new_message_int_->index= nu_points ;
            _new_message_int_->value= trainClass[nu_points] ;
            dsWriteBuffer(outputstream34P, (    _new_message_int_),
                sizeof( message_int));
            nu_points=(nu_points+1);
            temp_dimentions=0;
        }

    }
    dsWriteBuffer(outputstream10P, ( & nu_dimentions), sizeof(
        nu_dimentions));
    dsWriteBuffer(outputstream14P, ( & nu_points), sizeof(nu_points)
        );
    dsWriteBuffer(outputstream13P, ( & nu_points), sizeof(nu_points)
        );

}
else
{
    printf("Cannot open TrainFile: %s\n", trainFileName);
    exit(0);
}

fclose(Arq);
dsCloseOutputPort ( outputstream34P ) ;
dsCloseOutputPort ( outputstream9P ) ;
if (Arq=fopen(testFileName, "r+"))
{
    nu_test=0;
    temp_dimentions=0;
    for (i=0; ! feof(Arq); i=0)
    {
        line[0]='\0';
        fgets(line, 1024, Arq);
        l=strlen(line);
        //remove \n char {}
        if (line[(l-1)]=='\n')
```

```
            {
                l --;
            }

            line[l]='\0';
            if (l>0)
            {
                token=strtok(line, " ");
                for (i=0; token!=((void *)0); i=0)
                {
                    testData[nu_test][temp_dimentions]=atof(token);
                    temp_dimentions=(temp_dimentions+1);

                    token=strtok(((void *)0), " ");
                }

                if (temp_dimentions!=nu_dimentions)
                {
                    printf("Error, wrong number of dimensions on test file:
                        %s\n", testFileName);
                    exit(0);
                }
                _new_message_float_array_->index = nu_test ;
                memcpy(_new_message_float_array_->value , testData[nu_test
                    ],sizeof(float)*COL_SIZE );
                dsWriteBuffer(outputstream12P, (
                    _new_message_float_array_), sizeof( message_float_array
                    )); ;

                temp_dimentions=0;
                nu_test=(nu_test+1);
            }


        }
        dsWriteBuffer(outputstream6P, ( & nu_test), sizeof(nu_test));
        dsWriteBuffer(outputstream5P, ( & nu_test), sizeof(nu_test));

    }
    else
    {
        printf("Cannot open TestFile: %s\n", testFileName);
        exit(0);
    }
```

```
    fclose(Arq);
    dsCloseOutputPort ( outputstream12P ) ;
    for (i=0; i<ROW_SIZE; i ++ )
    {

        distance[i]=0;
        _new_message_float_->index = i ;
        _new_message_float_->value = distance[i] ;
        dsWriteBuffer(outputstream11P, (    _new_message_float_), sizeof(
            message_float)); ;
        dsWriteBuffer(outputstream32P, (    _new_message_float_), sizeof(
            message_float)); ;
    }
    dsCloseOutputPort ( outputstream11P ) ;
    dsCloseOutputPort ( outputstream32P ) ;
    for (i=0; i<ROW_SIZE; i ++ )
    {

        dataIndex[i]=0;
        _new_message_int_->index = i ;
        _new_message_int_->value = dataIndex[i] ;
        dsWriteBuffer(outputstream7P, (    _new_message_int_), sizeof(
            message_int)); ;
        dsWriteBuffer(outputstream33P, (    _new_message_int_), sizeof(
            message_int)); ;
    }
    dsCloseOutputPort ( outputstream7P ) ;
    dsCloseOutputPort ( outputstream33P ) ;

}









gettimeofday(&tend,NULL);
strcpy(instanceTime,elapsed_time(tbegin,tend));
dsWriteBuffer(outputstream31P, ( & instanceTime), sizeof(instanceTime)
    );

    for (i=0;i<ROW_SIZE;i++){
```

```
            free(trainData[i]);
        }
        free(trainData);
        for (i=0;i<ROW_SIZE;i++){
            free(testData[i]);
        }
        free(testData);
        free(distance);
        free(dataIndex);
        free (trainClass);


        return 0;
}

int finalizeFilter( )
{
    printf("stoping filter5\n");
    return 0;
}
```

B.3.2.12 filter5.h

```
#ifndef FILTER5_H_
#define FILTER5_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER5_H_*/
```

B.3.2.13 filter6.c

```
#include "FilterDev.h"

#include <stdio.h>

#include <stdlib.h>
```

```
#include <string.h>

#include "work.h"
#include "util.h"
#include "messages.h"
#include "filter6.h"
int __new_iteration__;
InputPortHandler inputstream6P;
InputPortHandler inputstream15P;
InputPortHandler inputstream16P;
InputPortHandler inputstream26P;
InputPortHandler inputstream27P;
InputPortHandler inputstream28P;
InputPortHandler inputstream29P;
InputPortHandler inputstream30P;
InputPortHandler inputstream31P;
int initFilter(void * work, int size)
{
   printf("Inicializing filter6\n");
   inputstream31P=dsGetInputPortByName("inputstream31P");
   inputstream30P=dsGetInputPortByName("inputstream30P");
   inputstream29P=dsGetInputPortByName("inputstream29P");
   inputstream28P=dsGetInputPortByName("inputstream28P");
   inputstream27P=dsGetInputPortByName("inputstream27P");
   inputstream26P=dsGetInputPortByName("inputstream26P");
   inputstream16P=dsGetInputPortByName("inputstream16P");
   inputstream15P=dsGetInputPortByName("inputstream15P");
   inputstream6P=dsGetInputPortByName("inputstream6P");
   return 0;
}

int processFilter(void * work, int size)
{
   float *finalDistance;
   int *testClass;
   int nu_test;
   int i;
   int j;

   printf("Processing filter6\n");
   struct timeval tbegin, tend;
   char instanceTime[1024];
   char outputFileName[256];
```

```
FILE * Arq;
InputPortHandler *timeStreams[6 ];
finalDistance = (float *)malloc(sizeof(float )*ROW_SIZE);
testClass = (int *)malloc(sizeof(int )*ROW_SIZE);


gettimeofday(&tbegin,NULL);


{
    dsReadBuffer(inputstream6P, ( & nu_test), sizeof(nu_test));

    dsReadBuffer(inputstream16P, ( testClass), sizeof(int)*ROW_SIZE);
    dsReadBuffer(inputstream15P, (  finalDistance), sizeof(float)*
        ROW_SIZE);
    for (i=0; i<nu_test; i ++ )
    {
        printf("Distance: %f\tClass: %d\n", finalDistance[i], testClass[
            i]);
    }

}

sprintf (outputFileName,"%s_%d","outputWriterFilter",(int)time(NULL));
Arq=fopen(outputFileName,"w+");
timeStreams[0] = & inputstream26P ;
timeStreams[1] = & inputstream27P ;
timeStreams[2] = & inputstream28P ;
timeStreams[3] = & inputstream29P ;
timeStreams[4] = & inputstream30P ;
timeStreams[5] = & inputstream31P ;
dsReadBuffer(*timeStreams[5], ( & instanceTime ), sizeof(instanceTime)
    );
fprintf(Arq,"Filter %d read TIME: %s\n",5,instanceTime);
for (i=0;i<5;i++){
    for (j=0;j<dsGetNumWriters(*timeStreams[i]);j++){
        dsReadBuffer(*timeStreams[i], ( & instanceTime ), sizeof(
            instanceTime));

        fprintf(Arq,"Filter %d instance %d TIME: %s\n",i,j,instanceTime)
            ;
    }
}
gettimeofday(&tend,NULL);
```

```
    fprintf(Arq,"Filter 6 TIME: %s\n",elapsed_time(tbegin,tend));
    fclose(Arq);

    return 0;
}

int finalizeFilter( )
{
    printf("stoping filter6\n");
    return 0;
}
```

B.3.2.14 filter6.h

```
#ifndef FILTER6_H_
#define FILTER6_H_
#include "FilterDev.h"

int initFilter(void *work, int size);

int processFilter(void *work, int size);

int finalizeFilter();

#endif /*FILTER6_H_*/
```

B.3.2.15 knn.c

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <string.h>

#include "work.h"
#include "void.h"
#include "util.h"
int main(int argc, char * * argv)
{
    char trainFileName[1024];
```

```
    char testFileName[1024];
    int k;
    if (argc<4)
    {
        printf("Usage: %s <train_file> <test_file> <k>\n", argv[0]);
        exit(0);
    }

    strcpy(trainFileName, argv[1]);
    strcpy(testFileName, argv[2]);
    k=atoi(argv[3]);
    /*
        Statement removed because pragma READ annotation found
        */
    Work *work = (Work *)malloc(sizeof(Work));
    work->k=k;
    strcpy (work->testFileName,testFileName);
    strcpy(work->trainFileName,trainFileName);
    char confFile[] = "./conf.xml";
    Layout *systemLayout = initDs(confFile, argc, argv);
    appendWork(systemLayout, (void *)work, sizeof(Work));
    finalizeDs(systemLayout);
    /*
#WHILE LOOP REMOVED - FILTERS INSERTED:  0 1 2 3 4
*/
    /*
        Statement removed because pragma WRITE annotation found
        */
    return 0;
}
```

B.3.2.16 labelFunc.c

```
int hash(char * label, int image)
{
int dest;
int * aux;
aux=((int * )( & label[0]));
dest=(( * aux)%image);
return dest;
}

void getLabel(void * msg, int size, char label[])
```

```
{
int * aux;
aux=((int * )( & label[0]));
( * aux)=( * ((int * )msg));
}
```

B.3.2.17 messages.h

```
#include "work.h"
struct message_int_t
{
   int index;
   int value;
};

typedef struct message_int_t message_int, * message_int_ptr;
struct message_float_t
{
   int index;
   float value;
};

typedef struct message_float_t message_float, * message_float_ptr;
struct message_float_array_t
{
   int index;
   float value[COL_SIZE];
};

typedef struct message_float_array_t message_float_array, *
    message_float_array_ptr;
```

B.3.2.18 work.h

```
#ifndef _WORK_H_
#define _WORK_H_
#define ROW_SIZE 10000
#define COL_SIZE 11
#define MAXLINESIZE 1024

struct _Work
{
```

```
    int k;
    char testFileName[1024];
    char trainFileName[1024];
};

typedef struct _Work Work, * Work_ptr;
#endif
```

B.3.2.19 conf.xml

```
<config>
   <hostdec>
      <host name="uzi01" mem="2048">
         <resource name="1"/>
         </host>
      <host name="uzi02" mem="2048">
         <resource name="2"/>
         </host>
      <host name="uzi03" mem="2048">
         <resource name="3"/>
         </host>
      <host name="uzi08" mem="2048">
         <resource name="4"/>
         </host>
      <host name="uzi09" mem="2048">
         <resource name="23"/>
         </host>
      <host name="uzi10" mem="2048">
         <resource name="6"/>
         </host>
      <host name="uzi11" mem="2048">
         <resource name="7"/>
         </host>
      <host name="uzi06" mem="2048">
         <resource name="8"/>
         </host>
      <host name="uzi14" mem="2048">
         <resource name="9"/>
         </host>
      <host name="uzi17" mem="2048">
         <resource name="10"/>
         </host>
            <host name="uzi18" mem="2048">
```

```
        <resource name="11"/>
        </host>
    <host name="uzi19" mem="2048">
        <resource name="12"/>
        </host>
    <host name="uzi20" mem="2048">
        <resource name="13"/>
        </host>
    <host name="uzi21" mem="2048">
        <resource name="14"/>
        </host>
    <host name="uzi22" mem="2048">
        <resource name="15"/>
        </host>
    <host name="uzi23" mem="2048">
        <resource name="16"/>
        </host>
    <host name="uzi24" mem="2048">
        <resource name="17"/>
        </host>
    <host name="uzi25" mem="2048">
        <resource name="18"/>
        </host>
    <host name="uzi28" mem="2048">
        <resource name="19"/>
        </host>
    <host name="uzi35" mem="2048">
        <resource name="20"/>
        </host>
    <host name="uzi32" mem="2048">
        <resource name="21"/>
        </host>
    <host name="uzi33" mem="2048">
        <resource name="22"/>
        </host>
    <host name="uzi05" mem="2048">
        <resource name="5"/>
        </host>

</hostdec>

<placement>
    <filter name="filter0" libname="filter0.so" instances="1">
     <instance demands="1" numinstances="1" />
```

```
    </filter>
    <filter name="filter4" libname="filter4.so" instances="1">
     <instance demands="2" numinstances="1" />
    </filter>
    <filter name="filter5" libname="filter5.so" instances="1">
     <instance demands="1" numinstances="1" />
    </filter>
    <filter name="filter6" libname="filter6.so" instances="1">
     <instance demands="2" numinstances="1" />
    </filter>
    <filter name="filter3" libname="filter3.so" instances="1">
     <instance demands="3" numinstances="1" />
    </filter>
  <filter name="filter1" libname="filter1.so" instances="1">
    <instance demands="4" numinstances="1" />
  </filter>

  <filter name="filter2" libname="filter2.so" instances="8">
    <instance demands="5" numinstances="1" />
    <instance demands="6" numinstances="1" />
    <instance demands="7" numinstances="1" />
    <instance demands="8" numinstances="1" />
    <instance demands="9" numinstances="1" />
    <instance demands="10" numinstances="1" />
    <instance demands="11" numinstances="1" />
    <instance demands="12" numinstances="1" />
  </filter>

  </placement>
  <layout>
    <stream>
      <from filter="filter0" port="outputstream26P" policy="broadcast"
          />
      <to filter="filter6" port="inputstream26P"/>
    </stream>
    <stream>
      <from filter="filter0" port="outputstream1P" policy="broadcast"
         />
      <to filter="filter2" port="inputstream1P"/>
    </stream>
    <stream>
      <from filter="filter0" port="outputstream0P" policy="broadcast"
         />
      <to filter="filter1" port="inputstream0P"/>
```

```
</stream>
<stream>
   <from filter="filter0" port="outputstream3P" policy="broadcast"
      />
   <to filter="filter3" port="inputstream3P"/>
</stream>
<stream>
   <from filter="filter0" port="outputstream2P" policy="broadcast"
      />
   <to filter="filter4" port="inputstream2P"/>
</stream>
<stream>
   <from filter="filter1" port="outputstream27P" policy="broadcast"
       />
   <to filter="filter6" port="inputstream27P"/>
</stream>
<stream>
   <from filter="filter1" port="outputstream23P"  policy="
      labeled_stream"  policyLib="labelFunc.so"  />
   <to filter="filter2" port="inputstream23P"/>
</stream>
<stream>
   <from filter="filter1" port="outputstream18P"   policy="
      labeled_stream"  policyLib="labelFunc.so" />
   <to filter="filter2" port="inputstream18P"/>
</stream>
<stream>
   <from filter="filter2" port="outputstream28P" policy="broadcast"
       />
   <to filter="filter6" port="inputstream28P"/>
</stream>
<stream>
   <from filter="filter2" port="outputstream22P" policy="
      labeled_stream"  policyLib="labelFunc.so"  />
   <to filter="filter1" port="inputstream22P"/>
</stream>
<stream>
   <from filter="filter2" port="outputstream17P" policy="
      labeled_stream"  policyLib="labelFunc.so"  />
   <to filter="filter1" port="inputstream17P"/>
</stream>
<stream>
   <from filter="filter2" port="outputstream24P" policy="broadcast"
       />
```

```
    <to filter="filter3" port="inputstream24P"/>
  </stream>
  <stream>
    <from filter="filter2" port="outputstream25P" policy="broadcast"
        />
    <to filter="filter3" port="inputstream25P"/>
  </stream>
  <stream>
    <from filter="filter3" port="outputstream29P" policy="broadcast"
        />
    <to filter="filter6" port="inputstream29P"/>
  </stream>
  <stream>
    <from filter="filter3" port="outputstream15P" policy="
       round_robin" />
    <to filter="filter6" port="inputstream15P"/>
  </stream>
  <stream>
    <from filter="filter3" port="outputstream16P" policy="
       round_robin" />
    <to filter="filter6" port="inputstream16P"/>
  </stream>
  <stream>
    <from filter="filter4" port="outputstream30P" policy="broadcast"
        />
    <to filter="filter6" port="inputstream30P"/>
  </stream>
  <stream>
    <from filter="filter4" port="outputstream19P" policy="broadcast"
        />
    <to filter="filter0" port="inputstream19P"/>
  </stream>
  <stream>
    <from filter="filter4" port="outputstream21P" policy="broadcast"
        />
    <to filter="filter3" port="inputstream21P"/>
  </stream>
  <stream>
    <from filter="filter4" port="outputstream20P" policy="broadcast"
        />
    <to filter="filter1" port="inputstream20P"/>
  </stream>
  <stream>
```

```
    <from filter="filter5" port="outputstream13P" policy="broadcast"
        />
    <to filter="filter1" port="inputstream13P"/>
</stream>
<stream>
    <from filter="filter5" port="outputstream14P" policy="broadcast"
        />
    <to filter="filter2" port="inputstream14P"/>
</stream>
<stream>
    <from filter="filter5" port="outputstream11P" policy="
        labeled_stream"  policyLib="labelFunc.so"  />
    <to filter="filter1" port="inputstream11P"/>
</stream>
<stream>
    <from filter="filter5" port="outputstream12P" policy="
        round_robin" />
    <to filter="filter1" port="inputstream12P"/>
</stream>
<stream>
    <from filter="filter5" port="outputstream9P" policy="round_robin
        " />
    <to filter="filter1" port="inputstream9P"/>
</stream>
<stream>
    <from filter="filter5" port="outputstream5P" policy="broadcast"
        />
    <to filter="filter0" port="inputstream5P"/>
</stream>
<stream>
    <from filter="filter5" port="outputstream7P" policy="
        labeled_stream"  policyLib="labelFunc.so"  />
    <to filter="filter1" port="inputstream7P"/>
</stream>
<stream>
    <from filter="filter5" port="outputstream31P" policy="broadcast"
        />
    <to filter="filter6" port="inputstream31P"/>
</stream>
<stream>
    <from filter="filter5" port="outputstream6P" policy="broadcast"
        />
    <to filter="filter6" port="inputstream6P"/>
</stream>
```

```
      <stream>
         <from filter="filter5" port="outputstream10P" policy="broadcast"
              />
         <to filter="filter1" port="inputstream10P"/>
      </stream>
      <stream>
         <from filter="filter5" port="outputstream32P" policy="
            labeled_stream"  policyLib="labelFunc.so"  />
         <to filter="filter2" port="inputstream32P"/>
      </stream>
      <stream>
         <from filter="filter5" port="outputstream33P" policy="
            labeled_stream"  policyLib="labelFunc.so"  />
         <to filter="filter2" port="inputstream33P"/>
      </stream>

      <stream>
         <from filter="filter5" port="outputstream34P" policy="
            round_robin" />
         <to filter="filter3" port="inputstream34P"/>
      </stream>
   </layout>
</config>
```

Referências Bibliográficas

[Agrawal et al., 1993] Agrawal, R.; Imielinski, T. & Swami, A. N. (1993). Mining association rules between sets of items in large databases. Em Buneman, P. & Jajodia, S., editores, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pp. 207--216, Washington, D.C.

[Aho et al., 2006] Aho, A. V.; Lam, M. S.; Sethi, R. & Ullman, J. D. (2006). *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley.

[Akimasa et al., 2009] Akimasa; Matsuzaki, K.; Hu, Z. & Takeichi, M. (2009). The third homomorphism theorem on trees: downward & upward lead to divide-and-conquer. Em *POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT sympos ium on Principles of programming languages*, pp. 177--185. ACM.

[Appel et al., 1998] Appel, A.; Davidson, J. & N., R. (1998). The zephyr compiler infrastructure. www.cs.virginia.edu/zephyr. Em *Proceedings of the International Conference on Supercomputing*.

[Beynon et al., 2000] Beynon, M.; Ferreira, R.; Kurc, T. M.; Sussman, A. & Saltz, J. H. (2000). Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. Em *IEEE Symposium on Mass Storage Systems*, pp. 119–134.

[Beynon et al., 2001] Beynon, M. D.; Kurc, T.; Catalyurek, U.; Chang, C.; Sussman, A. & Saltz, J. (2001). Distributed processing of very large datasets with datacutter. *Parallel Comput.*, 27(11):1457--1478.

[Chang et al., 1991] Chang, P. P.; Mahlke, S. A.; Chen, W. Y.; Warter, N. J. & mei W. Hwu, W. (1991). Impact: an architectural framework for multiple-instruction-issue processors. Em *ISCA '91: Proceedings of the 18th annual international symposium on Computer architecture*, pp. 266--275, New York, NY, USA. ACM Press.

[Cole, 1988] Cole, M. I. (1988). Algorithmic skeletons: a structured approach to the management of parallel computation. Research Monograph on Parallel and Distributed Computing.

[compiler development system, 2006] compiler development system, C. (2006). http://www.ace.nl/compiler/cosy.html.

[Consortium, 2005] Consortium, U. (2005). Upc language specifications v1.2. Relatório técnico LBNL-59208, George Washington University, Lawrence Berkeley National Laboratory.

[Cytron et al., 1989] Cytron, R.; Ferrante, J.; Rosen, B. K.; Wegman, M. N. & Zadeck, F. K. (1989). An efficient method of computing static single assignment form. Em *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 25--35, New York, NY, USA. ACM Press.

[da Mata et al., 2009] da Mata, L. L. P.; Pereira, F. M. Q. & Ferreira, R. A. C. (2009). Automatic parallelization of canonical loops. Em *XIII Brazilian Symposium on Programming Languages - SBLP'09*, Gramado-RS, Brazil.

[de Castro Santos, 2006] de Castro Santos, J. (2006). Particionamento semi-automático de reduções cíclicas para execução em anthill. Dissertação de mestrado, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais.

[Dean & Ghemawat, 2004] Dean, J. & Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. Em *OSDI*, pp. 137–150.

[Du & Agrawal, 2004] Du, W. & Agrawal, G. (2004). Language and compiler support for adaptive applications. Em *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, p. 29, Washington, DC, USA. IEEE Computer Society.

[Du & Agrawal, 2005] Du, W. & Agrawal, G. (2005). Filter decomposition for supporting coarse-grained pipelined parallelism. Em *ICPP*, pp. 539–546. IEEE Computer Society.

[Du et al., 2003] Du, W.; Ferreira, R. & Agrawal, G. (2003). Compiler support for exploiting coarse-grained pipelined parallelism. Em *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, p. 8, Washington, DC, USA. IEEE Computer Society.

[Ferreira et al., 2000] Ferreira, R.; Agrawal, G. & Saltz, J. (2000). Compiling object-oriented data intensive applications. Em *ICS '00: Proceedings of the 14th international conference on Supercomputing*, pp. 11--21, New York, NY, USA. ACM Press.

[Ferreira et al., 2005] Ferreira, R.; Jr., W. M.; Guedes, D. & Drumond, L. (2005). Anthill: A scalable run-time environment for data mining applications. Em *SBAC-PAD'05: The 17th International Symposium on Computer Architecture and High Performance Computing*, Rio de Janeiro, Brazil.

[Fireman et al., 2008] Fireman, D.; Teodoro, G.; Cardoso, A. & Ferreira, R. (2008). A reconfigurable run-time system for filter-stream applications. Em *SBAC-PAD'08: The 20th International Symposium on Computer Architecture and High Performance Computing*, Campo Grande, Brazil.

[for Analytical Experiments, 2006] for Analytical Experiments, S. A. S. C. (2006). http://www-ali.cs.umass.edu/scale/.

[Foster et al., 2002] Foster, I.; Kesselman, C.; Nick, J. & Tuecke, S. (2002). The physiology of the grid: An open grid services architecture for distributed systems integration. Em *Open Grid Service Infrastructure WG, Global Grid Forum*.

[Fraser & Hanson, 1991] Fraser, C. W. & Hanson, D. R. (1991). A retargetable compiler for ANSI C. Relatório técnico CS–TR–303–91, Princeton University, Princeton, N.J.

[Fraser & Hanson, 2003] Fraser, C. W. & Hanson, D. R. (2003). *A Retargetable C Compiler: Design and Implementation*. Addison-Wesley. FRA ch 03:1 1.Ex.

[Gibbons, 1996] Gibbons, J. (1996). The third homomorphism theorem. *J. Funct. Program.*, 6(4):657–665.

[Gupta & Amarasinghe, 2008] Gupta, R. & Amarasinghe, S. P., editores (2008). *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, Tucson, AZ, USA, June 7-13*. ACM.

[Gurd et al., 1985] Gurd, J. R.; Kirkham, C. C. & Watson, I. (1985). The manchester prototype dataflow computer. *Commun. ACM*, 28(1):34--52.

[Góes et al., 2005] Góes, L. F. W.; Stefani, I. G. A.; Ferreira, R. & Jr., W. M. (2005). Mapeamento de programas i3 para aplicações anthill paralelas de fluxos de dados baseadas em filtros. *VI Workshop em Sistemas Computacionais de Alto Desempenho WSCAD'2005*, pp. 145–152. Rio de Janeiro - RJ - Brasil.

[Hall et al., 2009] Hall, M.; Padua, D. & Pingali, K. (2009). Compiler research: the next 50 years. *Commun. ACM*, 52(2):60--67.

[Hall et al., 1995] Hall, M. H.; Amarasinghe, S. P.; Murphy, B. R.; Liao, S.-W. & Lam, M. S. (1995). Detecting coarse-grain parallelism using an interprocedural parallelizing compiler. Em *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, p. 49, New York, NY, USA. ACM Press.

[Hall et al., 1996] Hall, M. W.; Anderson, J.-A. M.; Amarasinghe, S. P.; Murphy, B. R.; Liao, S.-W.; Bugnion, E. & Lam, M. S. (1996). Maximizing multiprocessor performance with the SUIF compiler. *IEEE Computer*, 29(12):84–89.

[Han & Tseng, 1999] Han, H. & Tseng, C.-W. (1999). Improving compiler and run-time support for irregular reductions using local writes. Em *LCPC '98: Proceedings of the 11th International Workshop on Languages and Compilers for Parallel Computing*, pp. 181--196, London, UK. Springer-Verlag.

[Hofstee, 2005] Hofstee, H. P. (2005). Power efficient processor architecture and the cell processor. Em *HPCA*, pp. 258--262. IEEE Computer Society.

[internals Manual, 2005] internals Manual, G. (2005). http://gcc.gnu.org/onlinedocs/gccint/.

[Johnson et al., 2004] Johnson, T. A.; Lee, S. I.; Fei, L.; Basumallik, A.; Upadhyaya, G.; Eigenmann, R. & Midkiff, S. P. (2004). Experiences in using cetus for source-to-source transformations. Em *LCPC*, pp. 1–14.

[Johnston et al., 2004] Johnston, W. M.; Hanna, J. R. P. & Millar, R. J. (2004). Advances in dataflow programming languages. *ACM Comput. Surv.*, 36(1):1--34.

[Kongetira et al., 2005] Kongetira, P.; Aingaran, K. & Olukotun, K. (2005). Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29.

[Lattner, 2002] Lattner, C. (2002). LLVM: An Infrastructure for Multi-Stage Optimization. Dissertação de mestrado, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL. http://llvm.cs.uiuc.edu.

[Lattner & Adve, 2004] Lattner, C. & Adve, V. (2004). LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. Em *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California.

[Lee et al., 2003] Lee, S. I.; Johnson, T. A. & Eigenmann, R. (2003). Cetus - an extensible compiler infrastructure for source-to-source transformation. Em *LCPC*, pp. 539–553.

[Li et al., 2003] Li, X.; Jin, R. & Agrawal, G. (2003). A compilation framework for distributed memory parallelization of data mining algorithms. Em *17th International Parallel and Distributed Processing Symposium (IPDPS-2003)*, pp. 7--7, Los Alamitos, CA. IEEE Computer Society.

[Lim et al., 1999] Lim, A. W.; Cheong, G. I. & Lam, M. S. (1999). An affine partitioning algorithm to maximize parallelism and minimize communication. Em *International Conference on Supercomputing*, pp. 228–237.

[Macqueen, 1967] Macqueen, J. B. (1967). Some methods of classification and analysis of multivariate observations. Em *Proceedings of the Fifth Berkeley Symposium on Mathemtical Statistics and Probability*, pp. 281--297.

[mei Hwu, 2006] mei Hwu, W. (2006). Open impact compiler http://www.gelato.org/.

[Morita et al., 2007] Morita, K.; Morihata, A.; Matsuzaki, K.; Hu, Z. & Takeichi, M. (2007). Automatic inversion generates divide-and-conquer parallel programs. Em *PLDI*, pp. 146–155. ACM.

[Morrison, 1994] Morrison, J. P. (1994). *Flow-Based Programming*. Van Nostrand Reinhold.

[Multi-Processing, 1997] Multi-Processing, O. O. (1997). http://openmp.org.

[Necula et al., 2002] Necula, G. C.; McPeak, S.; Rahul, S. P. & Weimer, W. (2002). CIL: Intermediate language and tools for analysis and transformation of c programs. Em *CC '02: Proceedings of the 11th International Conference on Compiler Construction*, pp. 213--228, London, UK. Springer-Verlag.

[Parr & Quong, 1995] Parr, T. J. & Quong, R. W. (1995). ANTLR: A predicated-LL(k) parser generator. *Software: Practice and Experience*, 25(7):789--810.

[Ponnusamy et al., 1993] Ponnusamy, R.; Saltz, J. H. & Choudhary, A. N. (1993). Runtime compilation techniques for data partitioning and communication schedule reuse. Em *Supercomputing*, pp. 361–370.

[Rauchwerger & Padua, 1999] Rauchwerger, L. & Padua, D. A. (1999). The LRPD test: Speculative run-time parallelization of loops with privatization and reduction

parallelization. *IEEE Transactions on Parallel and Distributed Systems*, 10(2):160--??

[Remy & Vouillon, 1998] Remy, D. & Vouillon, J. (1998). Objective ML: An effective object-oriented extension to ML. *Theory and Practice of Object Systems*, 4(1):27–50.

[Ryoo et al., 2008] Ryoo, S.; Rodrigues, C. I.; Baghsorkhi, S. S.; Stone, S. S.; Kirk, D. B. & mei W. Hwu, W. (2008). Optimization principles and application performance evaluation of a multithreaded gpu using cuda. Em *PPoPP*, pp. 73--82. ACM.

[Smith, 1996] Smith, M. D. (1996). Extending SUIF for machine-dependent optimizations. Em *Proceedings of the First SUIF Compiler Workshop*, pp. 14--25.

[So et al., 1998] So, B.; Moon, S. & Hall, M. W. (1998). Measuring the effectiveness of automatic parallelization in suif. Em *ICS '98: Proceedings of the 12th international conference on Supercomputing*, pp. 212--219, New York, NY, USA. ACM Press.

[Spring et al., 2007] Spring, J. H.; Privat, J.; Guerraoui, R. & Vitek, J. (2007). Streamflex: high-throughput stream programming in java. Em *Conference on Object oriented programming systems and applications*, pp. 211--228. ACM.

[Steinhaus, 1956] Steinhaus, H. (1956). Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1:801--804.

[Sutherland, 1966] Sutherland, W. R. (1966). *The on-line graphical specification of computer procedures*. Tese de doutorado, Massachusetts Institute of Technology. Dept. of Electrical Engineerin.

[Teodoro et al., 2008] Teodoro, G.; Fireman, D.; Neto, D. O. G.; Jr., W. M. & Ferreira, R. (2008). Achieving multi-level parallelism in the filter-labeled stream programming model. Em *ICPP*, pp. 287–294. IEEE Computer Society.

[Whiting & Pascoe, 1994] Whiting, P. G. & Pascoe, R. S. V. (1994). A history of data-flow languages. *IEEE Ann. Hist. Comput.*, 16(4):38--59.

[Witten & Frank, 2002] Witten, I. H. & Frank, E. (2002). Data mining: practical machine learning tools and techniques with Java implementations. *ACM SIGMOD Record*, 31:76--77.

[Witten & Frank, 2005] Witten, I. H. & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edição.

[Wolfe, 1996] Wolfe, M. (1996). *High Performance Compilers for Parallel Computing*. Addison-Wesley.

[Yelick et al., 1998] Yelick, K.; Semenzato, L.; Pike, G.; Miyamoto, C.; Liblit, B.; Krishnamurthy, A.; Hilfinger, P.; Graham, S.; Gay, D.; Colella, P. & Aiken, A. (1998). Titanium: A high-performance Java dialect. Em ACM, editor, *ACM 1998 Workshop on Java for High-Performance Network Computing*, New York, NY 10036, USA. ACM Press.