FUSÃO DE DADOS EM GAMMA

Rui Rodrigues de Mello Junior

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Felipe Maia Galvão França
Gabriel Antoine Louis Paillard

Rio de Janeiro
Maio de 2015

FUSÃO DE DADOS EM GAMMA

Rui Rodrigues de Mello Junior

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Gabriel Antoine Louis Paillard, D.Sc.

Prof. Mário Roberto Folhadela Benevides, Ph.D.

Prof. Fábio Protti, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2015

Mello Junior, Rui Rodrigues de

Fusão de Dados em Gamma/Rui Rodrigues de Mello Junior. – Rio de Janeiro: UFRJ/COPPE, 2015.

XVI, 94 p.: il.; 29, 7cm.

Orientadores: Felipe Maia Galvão França

Gabriel Antoine Louis Paillard

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2015.

Referências Bibliográficas: p. 85 – 89.

1. Gamma. 2. Fusão de Dados. 3. Computação Paralela. I. França, Felipe Maia Galvão *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

"Uma mente que se abre a uma nova ideia jamais voltará ao seu tamanho original"
(Albert Einstein)

Dedico este trabalho aos meus pais e meu irmão pela educação, caráter e senso de responsabilidade transmitidos, a minha esposa Priscila, pelo companheirismo, paciência e dedicação demonstrados durante o período do curso e a meu filho Bernardo, por ser a razão da minha existência. Sem o apoio total e irrestrito de todos vocês, nada teria sido possível.

Agradecimentos

Agradeço primeiramente a Deus pela minha vida, dos meus familiares e amigos, e por sua infinita misericórdia, permitir que chegasse até aqui.

À minha esposa Priscila, por todo companheirismo, amor e principalmente por ter me dado o melhor presente da minha vida: nosso filho Bernardo, o qual faz a minha vida ter sentido e razão de ser. Amo vocês.

Aos meus pais, Sueli e Ruy, por toda educação e amor irrestritos. Sem o apoio e dedicação de vocês eu nada seria. Obrigado por me fazerem ser o homem que sou. Não somente esta, mas todas as conquistas da minha vida devo a vocês.

Ao meu irmão Rafael por todo o companheirismo, apoio, dedicação e amizade, hoje e sempre. OSS!

A todos os meus familiares que acreditaram no meu potencial e me incentivaram a buscar sempre mais.

Ao Shihan Lirton dos Reis Monassa (*in memorian*), por ter contribuído para a formação do meu caráter e por ter me ensinado a criar o intuito do esforço, dentre tantas lições. OSS!

À Rubens Pailo, grande amigo, companheiro de mestrado e de trabalho, por todas as dúvidas esclarecidas, sugestões e apoio, desde a fase em que cursamos as disciplinas até o momento da defesa desta dissertação. Muito obrigado.

Aos meus chefes e encarregados do Instituto de Pesquisas da Marinha, especialmente ao CMG (RM1) Andrada pelo grande incentivo e apoio, CMG(EN) Sineiro e CC(EN) André Chaves por concordarem e apoiarem a realização deste curso de mestrado.

À Rogério Perroti, por toda a paciência nas explicações sobre o PPDE e demais conceitos de fusão de dados além do grande incentivo durante nossas conversas.

Aos amigos da Equipe do CLP, Maurício, Fábio e Vitor Ribas pelo apoio e compreensão nos momentos em que precisei me ausentar e pelo incentivo em concluir este curso.

Á Alessandro Marino e Victor Blunk por todo apoio e motivação nos momentos em que a carga de trabalho parecia intransponível. E a todos do projeto SCM pela lealdade e amizade.

Aos demais colegas do Grupo de Sistemas Digitais que incentivaram e contribuíram para a realização deste trabalho.

Ao Programa de Engenharia de Sistemas e Computação (PESC/COPPE/UFRJ) por todo o suporte fornecido que me permitiu desenvolver este trabalho. Em especial gostaria de agradecer aos professores Claudio Amorim, Valmir Barbosa, Ricardo Farias, Leandro Marzulo, Rosa Leão, Jayme Szwarcfiter, e Myrian Costa, pelos conhecimentos transmitidos nas disciplinas cursadas durante o mestrado. Agradeço também aos funcionários do PESC pelo suporte e apoio prestados.

Aos professores Felipe França e Gabriel Paillard, pela orientação deste trabalho e por estarem sempre dispostos a ajudar. Obrigado pelo companheirismo, amizade e paciência que contribuíram para o meu aprimoramento profissional e pessoal.

Aos amigos do PESC: Rafael, Edilson, Luneque e Israel pelo companheirismo e motivação do dia a dia.

Ao prestimoso apoio do CENAPAD-UFC, por apoiar a realização deste trabalho, permitindo a execução dos experimentos em seu supercomputador.

À Universidade Católica de Petrópolis e seus professores, pela base sólida de minha formação.

Ao professor e orientador de graduação José Abdalla Helayel-Neto por despertar em mim a paixão pela ciência.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

FUSÃO DE DADOS EM GAMMA

Rui Rodrigues de Mello Junior

Maio/2015

Orientadores: Felipe Maia Galvão França
Gabriel Antoine Louis Paillard

Programa: Engenharia de Sistemas e Computação

Em 1986 foi proposto um formalismo para especificação de programas baseado na reescrita paralela de multiconjuntos, chamado de Gamma. Tal modelo pode ser encarado como uma máquina química abstrata, pois fornece uma metáfora de reações químicas ao passo que todas as ações envolvidas (reações químicas) ocorrem livremente e de maneira natural sobre uma base de dados única (solução química) chamada de multiconjunto. A área de fusão de dados tem dispendido bastante esforço em pesquisa, no meio militar e civil. A Marinha do Brasil desenvolveu um sistema de navegação que utiliza um método de fusão de dados para acompanhamento de contatos, o PPDE, que processa grafos em dois estágios visando fornecer a melhor hipótese de caminho de um alvo e, consequentemente, determinar a posição do mesmo no instante de tempo corrente. Neste trabalho é realizado o projeto e implementação de um algoritmo de fusão de dados para acompanhamento de contatos baseado em uma solução de grafos em dois estágios utilizando o modelo computacional Gamma. A solução proposta consiste na primeira implementação paralela do método PPDE. Para tanto, foram empregadas três implementações de Gamma, dentre as quais duas exploraram os recursos de um ambiente de hardware paralelo. Assim, o algoritmo proposto foi analisado do ponto de vista do paralelismo explorado e por fim foi realizada uma análise de desempenho deste algoritmo nas três implementações de Gamma utilizadas. Pretende-se com este trabalho fornecer uma implementação de um problema real sobre o formalismo Gamma, o que contribui para as implementações do modelo computacional Gamma, uma vez que possibilita a realização de análises de desempenho destas implementações e fornece algumas sugestões de possíveis melhorias. Além disso, o trabalho contribui para o método PPDE uma vez que fornece subsídios para a paralelização do primeiro estágio.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DATA FUSION IN GAMMA

Rui Rodrigues de Mello Junior

May/2015

Advisors: Felipe Maia Galvão França
Gabriel Antoine Louis Paillard

Department: Systems Engineering and Computer Science

In 1986, was proposed a formalism to program specification based on pararallel multiset rewriting, called Gamma. This model can be seen as an abstract chemical machine, it provides a metaphor for chemical reactions, once all of actions involved (chemical reactions) occur freely and naturally, on a single data base (chemical solution) called multiset. The data fusion area has expended great effort in research in the military and civilian areas. The Brazilian Navy developed a navigation system that uses a target tracking data fusion method, called PPDE, which uses a processing graph in two stages to provide the best hypothesis for the target path and thus determine the position of this target at the current time. In this work we carried out designing and implementing a target tracking data fusion algorithm based on a two stages graph solution using the computational model Gamma. The proposed solution is the first parallel implementation of the method PPDE. For this, we employed three Gamma implementations, where two of them exploited the resources of a parallel hardware environment. Thus, the proposed algorithm was evaluated from the parallelism exploited and finally was carried out a performance analysis of this algorithm in the three Gamma implementations used. The aim of this study is to provide an implementation on a real problem based in Gamma formalism, which contributes to the implementations of the Gamma computational model, since it enables the performance analyzes of these implementations and provides some suggestions for possible improvements. In addition, this work contributes to the PPDE method since it provides subsidies for the parallelization of the first stage.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

Apesar do grande avanço no que diz respeito à tecnologia computacional, os computadores sequenciais aproximam-se do seu limite físico [1, 2]. Por exemplo, o aumento de desempenho dos processadores esbarra na barreira física de capacidade de dissipação de calor e energia, entre outas limitações. Assim, a computação paralela surge como uma potencial alternativa para superar tais limitações. Entretanto, programar de forma paralela pode não ser uma tarefa simples. Realizar as atividades de distribuição e controle de tarefas entre processadores, controle de carga, gerenciamento de troca de mensagens entre os mesmos, além de realizar toda a modelagem do problema de forma a extrair o máximo de paralelismo, tornam bastante árdua e complexa a tarefa de programar de maneira paralela.

Neste contexto surge o Gamma (**G**eneral **A**bstract **M**odel for **M**ultiset m**A**nipulation), um formalismo para especificação de programas baseado na reescrita paralela de multiconjuntos. O conceito foi inicialmente proposto em 1986 por BANÂTRE e LE MÉTAYER [3]. Em Gamma, a única base de dados existente é o multiconjunto, onde todos os dados utilizáveis são representados por elementos deste. O modelo de execução é não determinístico, uma vez que os elementos do multiconjunto podem reagir livremente e de maneira naturalmente paralela, tornando transparente ao programador detalhes inerentes à implementação do paralelismo. Gamma baseia-se em uma metáfora de reações químicas onde podemos enxergar o multiconjunto como uma solução química composta por diversas moléculas (elementos). Diversas ações (reacões químicas) ocorrem especificadas pelas correspondentes condições (condição de reação). Dessa forma, Gamma coloca-se como importante ferramenta para superar os limites tecnológicos existentes por consistir em um modelo conceitualmente paralelo e que permite expressar problemas de uma maneira simples e natural.

Como exemplo de área de atuação onde o paralelismo coloca-se como solução viável para problemas complexos, podemos citar a fusão de dados aplicada ao acompanhamento de contatos (ou alvos) no meio militar naval. Atualmente uma das principais atribuições das forças navais é a patrulha de suas costas. Para isto, o poder naval utiliza-se do processamento de diversas informações recebidas por inúmeros sensores distribuídos de maneira estratégica. Do ponto de vista tático, situação semelhante ocorre, onde um navio de guerra recebe informações de diversos sensores sobre possíveis alvos inimigos ou algum objeto de interesse. Assim, tanto em sistemas de vigilância quanto em sistemas táticos de navegação, um método para fusão de dados de múltiplos sensores torna-se de importância estratégica para qualquer Marinha.

Especificamente a Marinha do Brasil, possui um sistema de navegação chamado CISNE (Centro de Integração de Sensores e Navegação Eletrônica). Neste sistema encontra-se implementado um método de fusão de dados para acompanhamento de contatos conhecido pela sigla PPDE (Par de Plots em Dois Estágios). O PPDE é um método baseado em grafos proposto por BARBOSA [4], onde a fusão de dados é aplicada para utilização de um conjunto de informações, muitas vezes redundantes, de diversos sensores com o objetivo de conhecer o caminho percorrido por um alvo no cenário tático e com isso ter condição de identificar em tempo real a localização atual de tal objeto de interesse. O PPDE foi inspirado no PP (Par de Plot) [5], método para fusão de dados para acompanhamento de contatos também baseado em grafos.

No PP, todo o processamento dos dados é realizado em um único estágio onde os dados são inicialmente adquiridos pelos sensores e, posteriormente, tornar-se-ão vértices para um único grafo. Após isto são formadas arestas segundo critérios específicos do método baseados na consistência cinemática do alvo e, tendo o grafo sido formado, um algoritmo de otimização (Algoritmo de Dantzig) é aplicado visando a busca da melhor hipótese de caminho para o alvo em questão.

Já no PPDE, o algoritmo é dividido em dois estágios, onde no primeiro deles um grafo é formado para cada conjunto de dados recebidos por cada sensor. Dessa forma, no primeiro estágio, a partir dos grafos existentes para cada sensor, são escolhidas as melhores hipóteses de caminho para o alvo em questão, através da aplicação do mesmo algoritmo de otimização utilizado no PP (algoritmo de Dantzig). Ao final deste primeiro estágio, tem-se um conjunto de pontos (plots) que correspondem aos pontos que pertencem ao melhor caminho escolhido em cada grafo e, estes dados dão início ao segundo estágio gerando um grafo cujos vértices são os melhores plots escolhidos no estágio inicial. Ao final do segundo estágio, onde Dantzig é novamente aplicado, temos a melhor hipótese de caminho percorrido para o alvo. Em resultados experimentais, o PPDE obteve desempenho superior ao PP [4].

Tanto o PPDE quanto o PP objetivam identificar quem é o alvo no instante de tempo atual, utilizando para isto informações de plots (dados recebidos dos sensores que podem indicar um alvo ou ruído) recebidos no passado. Estes "plots do passado" mais os plots do instante de tempo atual irão compor o grafo. Uma vez escolhido o ponto que representa a posição atual do alvo, ele juntamente com os N-1 pontos escolhidos, representarão o caminho mais provável do alvo, no momento corrente, podendo ser alterado no próximo instante de tempo. Esta escolha baseia-se em critérios que permitirão a definição do caminho mais provável do alvo, visto que uma menor variação destes parâmetros, no caso parâmetros cinemáticos, poderá indicar a melhor hipótese de caminho que representará o deslocamento do alvo.

1.2 Motivação

A divisão em dois estágios proposta pelo PPDE torna o algoritmo potencialmente paralelizável, a medida em que o processamento do grafo formado para cada sensor, no primeiro estágio, pode ser realizado simultaneamente. Na revisão da literatura realizada por ocasião deste trabalho, não foi encontrado nenhum estudo que forneça a paralelização do PPDE. Porém, o investimento em paralelismo pode trazer inúmeras vantagens relacionadas ao desempenho do método, segundo BARBOSA [4]. Dessa forma, a inspiração para este trabalho surgiu da percepção desta possibilidade de paralelização do primeiro estágio do algoritmo.

O modelo computacional Gamma fornece a possibilidade de expressar problemas de uma maneira simples e naturalmente paralela. Ou seja, detalhes de implementação do paralelismo são transparentes para o programador. Gamma fornece um estilo de programação mais robusto que os paradigmas tradicionais, do ponto de vista do desenvolvimento de programas paralelos, uma vez que não existe a preocupação com restrições de sequencialidade.

Diante deste potencial de Gamma em expressar problemas com características de paralelismo e da necessidade de paralelização do PPDE, surge a motivação deste trabalho que foi expressar o método de fusão de dados para acompanhamento de contatos implementado no CISNE (o PPDE) segundo o formalismo Gamma. Além disso, a escolha de Gamma foi motivada pela possibilidade de investir em um modelo onde a exclusão de elementos indesejáveis (eliminação ruídos) pudesse ser realizada através de refinamentos (execução de reações) da base de dados (multiconjunto) de forma natural.

Por fim, torna-se motivante a expressão de um problema complexo segundo o formalismo Gamma pela possibilidade de contribuir com a validação das três implementações utilizadas. Além disso, expressar o problema de fusão de dados proposto de maneira clara e concisa pode ser considerada uma vantagem por si só se compa-

rada a forma complexa de expressar o mesmo problema de acordo com programas derivados de linguagens imperativas.

1.3 Objetivos e Contribuições

Para fins deste trabalho, foram utilizadas algumas implementações do formalismo Gamma. A primeira delas, chamada *Gamma-Sequencial*, utiliza somente um processador para a execução das reações envolvidas e não permite a execução em um hardware paralelo. Já a implementação *Gamma-MPI* foi expandida a partir da implementação anterior para executar em um hardware paralelo onde utiliza-se uma interface de troca de mensagens MPI (Message Passing Interface) [6]. Estas duas implementações foram desenvolvidas por Juarez Muylaert e Simon Gay. Já a terceira implementação utilizada neste trabalho, *Gamma-GPU* foi desenvolvida por DE ALMEIDA [7] e consiste em uma extensão de *Gamma-MPI* porém para execução em um ambiente de hardware paralelo com suporte a GPU.

Dessa forma, através da utilização das implementações de Gamma acima mencionadas, o presente trabalho possui os seguintes objetivos específicos:

- Fornecer uma implementação do algoritmo de Pares de Plots em Dois Estágios (PPDE) segundo o formalismo Gamma;
- Fornecer a primeira implementação paralela do método PPDE;
- Abordar algumas formas de explorar o paralelismo da solução proposta, segundo o formalismo Gamma; e
- Realizar a análise de desempenho das três implementações de Gamma utilizadas perante a implementação do algoritmo de fusão de dados para acompanhamento de contatos sugerido.

A implementação do algoritmo proposto nesta dissertação contribui sobremaneira para o próprio modelo computacional Gamma, uma vez que existem poucas implementações de problemas reais expressos no formalismo Gamma. Dentre os problemas reais implementados em Gamma, podemos citar alguns problemas de processamento de imagens [8, 9]. Assim, uma vez desenvolvido o algoritmo de fusão de dados para acompanhamento de contatos em Gamma, este poderá contribuir para o aperfeiçoamento das implementações de Gamma existentes pela complexidade e quantidade de reações envolvidas na solução. Tendo em vista esta oportunidade de contribuição, será realizada uma análise de desempenho destas implementações frente ao nosso algoritmo a partir de diferentes entradas.

Além disso, o trabalho contribui para o estudo do próprio método PPDE de forma inovadora, uma vez que não existe nenhuma implementação paralela para o

método citado, algo que traria enormes benefícios em desempenho comparado ao PP, segundo informações dos autores do método. Dessa forma, este trabalho seria a primeira implementação paralela do PPDE.

1.4 Estrutura do Texto

No segundo capítulo deste trabalho iremos abordar os detalhes inerentes ao formalismo Gamma, iniciando pelos principais conceitos envolvidos. O estilo de programação Gamma será explicitado com alguns exemplos de programas típicos. Após isto, veremos alguns padrões de transformações que ocorrem no multiconjunto, denominados esquemas de programação. Por fim, abordaremos o estado da arte em Gamma, onde serão mencionadas as principais extensões, aplicações e implementações de Gamma.

O Capítulo três é dedicado à Fusão de Dados para Acompanhamento de Contatos. Iniciaremos com uma introdução e a conceituação de fusão de dados, continuando pelos níveis de fusão de dados fornecidos pela JDL (*U.S. Joint Directors of Laboratories, Data Fusion Group*) e características da abordagem multisensor. Após isso, veremos algumas aplicações militares e não militares, descrição de um acompanhamento (*track*), passando pelos conceitos de associação e estimativa, e, chegando ao MHT (*Multiple Hypothesis Tracking*). Este capítulo é encerrado pela abordagem dos métodos baseados em grafos, destacando o PP (Par de Plots) e o PPDE (Par de Plots em Dois Estágios).

Após a apresentação do modelo de programação Gamma e do problema que motivou este trabalho, veremos maiores detalhes sobre a solução proposta no quarto capítulo. Antes de abordarmos o algoritmo proposto, iniciaremos o Capítulo quatro pelas implementações de Gamma empregadas, passando pelas suas principais características e sintaxe suportada. Em seguida veremos algumas considerações sobre PP e PPDE, onde serão abordados alguns detalhes de implementação destes métodos. Somente então abordaremos a solução desenvolvida, onde veremos detalhes sobre o multiconjunto e sobre o algoritmo proposto. Ao apresentarmos o algoritmo em si, abordaremos detalhes sobre as condições de parada que levaram a utilização de um grafo completo, cálculo do melhor caminho no grafo e um detalhamento das reações envolvidas. Continuando este capítulo, verificaremos a implementação dos estágios, detalhes sobre a dinâmica iterativa, as três abordagens sugeridas (Sequencial, Paralela e Híbrida), algumas dificuldades encontradas e finalizaremos o quarto capítulo por alguns trabalhos correlatos.

O quinto capítulo refere-se a validação da solução. Iniciaremos pela descrição do ambiente de testes utilizado, passando por detalhes dos casos de teste executados e chegando ao resultados obtidos onde veremos os casos de teste referentes à

análise de detalhes inerentes ao paralelismo da solução e outros experimentos sobre o desempenho das três implementações utilizadas. Aqui os dados obtidos serão apresentados por uma série de tabelas e gráficos que serão discutidos na última seção deste capítulo.

Por fim, o Capítulo seis aborda a conclusão do trabalho, onde constam as considerações finais, lições aprendidas e trabalhos futuros.

Capítulo 2

Gamma

O presente capítulo inicia a fundamentação teórica deste trabalho que se baseia no modelo computacional Gamma e na fusão de dados aplicada ao acompanhamento de alvos radar. Inicialmente trataremos do modelo computacional Gamma, que foi inicialmente proposto em 1986 como um formalismo para a especificação de programas baseado na metáfora de reações químicas e que utiliza a reescrita paralela de multiconjuntos como veremos nas seções a seguir.

Dessa forma, o objetivo deste capítulo é introduzir os principais conceitos inerentes ao modelo computacional em questão, abordar detalhes do estilo de programação inerente ao Gamma e finalizar elencando os principais trabalhos e aplicações existentes que utilizaram Gamma como base.

2.1 Um Modelo Baseado em Reações Químicas

Segundo HENNESSY e PATTERSON [1], a tecnologia computacional experimentou incríveis progressos nos últimos 65 anos, desde que o primeiro computador de propósito geral foi desenvolvido. Apesar do ritmo surpreendente no qual esta tecnologia se desenvolve, dobrando sua capacidade a cada 18 meses, segundo a revisão da Lei de Moore de 1975, atualmente os computadores sequenciais aproximam-se de seu limite físico. Neste contexto, pode-se citar três importantes limitações ao aumento de desempenho [2]:

Power Wall: onde o aumento do desempenho dos processadores encontra-se limitado pela capacidade de dissipação de calor e energia;

Memory Wall: a eficiência das operações de transferência de dados entre memória e processador limitam o incremento de desempenho computacional; e

Frequency Wall: A frequência do cristal oscilador que gera os ciclos de operação (*clocks*) possui a barreira física da velocidade da luz, a qual está próxima de ser alcançada. Atualmente os processadores atingem frequências de ciclos de operação na casa dos $5GHz$, como por exemplo o *AMD FX-9590*.

Dessa forma, a computação paralela surge como um dos mecanismos propostos para superar as limitações descritas anteriormente, através de estudos para explorar o paralelismo a nível de threads (*TLP - Thread-Level Parallelism*), instruções (*ILP - Instruction-Level Parallelism*) e dados (*DLP - Data-Level Parallelism*) [1].

Neste contexo surge o GAMMA, acrônimo para **G**eneral **A**bstract **M**odel for **M**ultiset m**A**nipulation, inicialmente proposto por BANÂTRE e LE MÉTAYER [3] em 1986. Gamma pode ser definido como um formalismo para especificação de programas, baseado na reescrita paralela de multiconjuntos. Assim, a principal característica de um programa em Gamma é que este consiste em um modelo de execução não determinístico, uma vez que os elementos deste multiconjunto podem interagir livremente e de forma naturalmente paralela. Portanto, Gamma propõe um formalismo que possibilita a abstração dos detalhes que tornam difícil a tarefa de desenvolver programas em linguagens de programação paralelas clássicas ou tradicionais.

O modelo computacional Gamma baseia-se em uma metáfora de reações químicas, onde existe uma base de dados única, chamada de multiconjunto (solução química) composta por diversos elementos (moléculas). Diversas ações (reações químicas) são especificadas para serem executadas sobre os elementos do multiconjunto, de acordo com uma série de condições (condição de reação).

2.2 Programação Gamma

Veremos a seguir detalhes inerentes à programação Gamma, abordando o estilo e alguns esquemas de programação em Gamma.

2.2.1 Estilo de Programação

Um programa típico em Gamma consiste, basicamente, em pares de funções formadas por condições/ações, executados sobre os elementos do multiconjunto, o qual constitui a única base de dados onde os elementos que o compõem reagem livremente. A execução ocorre através da modificação do multiconjunto pela exclusão, inclusão e transformação dos elementos existentes. Neste modelo, o fim da computação ocorre quando um estado estável é alcançado, o que corresponde a um estado onde todas as reações terminaram suas execuções, ou seja, nenhuma reação consegue mais cumprir as condições para reagir.

Esta estrutura de controle que torna possível a execução de um programa segundo o paradigma Gamma é o operador Γ, que, conforme [10], pode ser formalmente definido por:

$$\begin{aligned} &\Gamma((R_1, A_1), ..., (R_m, A_m))(M) = \\ &if\ \forall\ i \in [1, m], \forall\ x_1, ..., x_n \in M, \neg R_i(x_i, ..., x_n) \\ &\qquad then\ M \\ &else\ let\ x_1, ..., x_n \in M, let\ i \in [1, m]\ such\ that\ R_i(x_i, ..., x_n)\ in \\ &\qquad \Gamma((R_1, A_1), ..., (R_m, A_m))((M - x_1, ..., x_n) + A_i(x_1, ..., x_n)) \end{aligned} \tag{2.1}$$

Os pares de funções (R_i, A_i), são aplicadas ao multiconjunto (M) e especificam as reações a serem utilizadas e suas condições para execução. A aplicação do par (R_i, A_i) em M resulta em substituir em M um subconjunto de elementos $(x_1, ..., x_n)$ tal que $R_i(x_1, ..., x_n)$ seja verdadeiro pelos elementos de $A_i(x_1, ..., x_n)$. Caso nenhum elemento satisfaça a condição de reação, o resultado da computação é o próprio M inicial. Caso contrário, o resultado é o multiconjunto M subtraído do subconjunto de elementos $(x_1, ..., x_n)$ acrescido do subconjunto especificado pela condição de reação $A_i(x_1, ..., x_n)$, ou seja, $((M - \{x_1, ..., x_n\}) + A_i(x_1, ..., x_n))$. Enquanto existirem condições verdadeiras para a execução das reações existentes, estas irão sendo executadas (reagindo) e consequentemente transformando o multiconjunto. É importante perceber que se uma ou mais condições de reações forem satisfeitas para alguns subconjuntos do multiconjunto simultaneamente, a decisão de qual reação irá executar ocorre de maneira não determinística, uma vez que estas reações poderão ser executadas independentemente e simultaneamente, o que confere ao modelo computacional Gamma uma característica naturalmente paralela [10].

Agora veremos um exemplo do estilo de programação Gamma. Tomemos como objeto de estudo o trecho de código abaixo, que corresponde a um programa em Gamma que encontra o menor elemento de um multiconjunto:

$$\begin{aligned} Menor(M) = \Gamma(R, A)\ \ (M)\ \ where \\ R(x, y) = x \geq y \\ A(x, y) = y \end{aligned} \tag{2.2}$$

Aqui, ao comparar elementos dois a dois (x, y), caso o elemento x seja maior ou igual a y, x é retirado do multiconjunto e y é mantido. Caso contrário $(x < y)$, nenhuma ação é executada no multiconjunto.

Imaginemos o seguinte multiconjunto $\{2, 10, 5, 6, 3\}$. Uma possível execução para este multiconjunto seria: após a escolha dos elementos $(10, 5)$, como par (x, y), o elemento 10 seria excluído e o 5 mantido no multiconjunto, pois $x \geq y$, tendo como resultado parcial o multiconjunto $\{2, 5, 6, 3\}$. Da mesma forma ao selecionar $(6, 5)$, teríamos $\{2, 5, 3\}$. Agora, caso o par $(2, 5)$ tenha sido escolhido, nenhuma operação seria realizada sobre o multiconjunto, uma vez que a condição de reação $(x \geq y)$, não foi satisfeita. Após a seleção dos elementos $(5, 2)$ o multiconjunto seria transformado para $\{2, 3\}$ e, finalmente, seria reduzido a $\{2\}$, quando o par $(3, 2)$ fosse utilizado como par (x, y). Conforme mencionado anteriormente, a iteração entre os elementos é livre e ocorre paralelamente, de forma que diversos pares (x, y), poderiam ter sido comparados ao mesmo momento. A Figura 2.1, ilustra a execução do exemplo acima.

Já o exemplo constante em [11] e descrito a seguir, demonstra de uma maneira elegante e concisa um programa em Gamma que se propõe a encontrar os números primos menores ou iguais a N:

$$\begin{aligned} Prime(M) = \Gamma(R, A)\ (\{2, ..., N\})\ \ where \\ R(x, y) = multiple(x, y) \\ A(x, y) = y \end{aligned} \tag{2.3}$$

A função *multiple* é verdadeira se e somente se x for múltiplo de y. Portanto, caso tal condição seja satisfeita, x é retirado do multiconjunto. Assim, números primos não serão descartados e, ao final da execução, restarão somente os números primos menores ou iguais a N, no multiconjunto $\{2, ..., N\}$.

2.2.2 Esquemas de Programação

Como vimos nas seções anteriores, o multiconjunto é a única representação dos dados em Gamma. Este pode ser formado por elementos que possuem valores simples ou compostos, como por exemplo tuplas [1]. Dessa forma, o resultado da computação é obtido através de operações sobre esta base de dados. Tais operações transformam o multiconjunto de acordo com um dos métodos citados a seguir.

Em [9], os autores mencionam que os resultados das operações em um multiconjunto são alcançados através dos métodos de Relaxamento (*Relaxation*), Expansão

[1] No Capítulo 4 veremos maiores detalhes sobre a representação de dados em um multiconjunto, segundo uma implementação específica de Gamma.

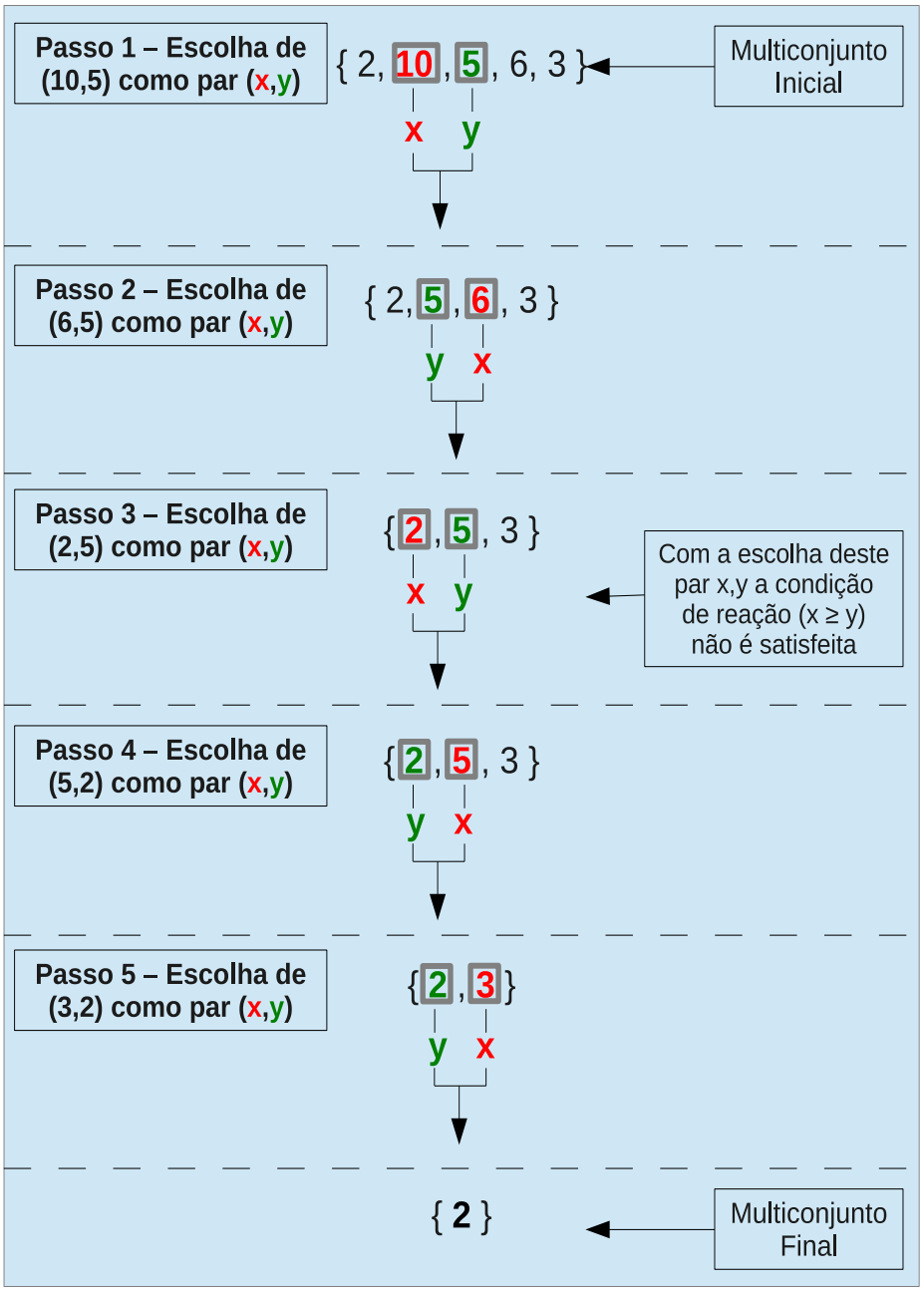

Figura 2.1: Exemplo da execução de um programa em Gamma

(*Data Expansion*) e Redução de Dados (*Data Reduction*). Relaxamento é um método utilizado para solucionar sistemas de equações através de iterações. Aqui, o multiconjunto vai passando por transformações a cada iteração correspondendo a eventuais resultados preliminares. A cada iteração os elementos indesejáveis vão sendo excluídos até que se alcance um resultado final. Expansão de dados é uma técnica utilizada neste modelo computacional que corresponde a decompor os elementos iniciais em elementos indivisíveis. Em outras palavras, o multiconjunto tem sua quantidade de elementos incrementada. Já a redução seria a operação antagônica à expansão. Ou seja, reduzir o multiconjunto a um único elemento, como resultado das reações executadas. Em [9], os autores ilustram estas técnicas com um exemplo

que calcula o enésimo termo de uma série de Fibonacci. Para tanto um número inicial N é decomposto em outros números (Expansão), que são somados para produzir o resultado esperado (Redução).

Em [8] e [12] são mencionados os *TROPES*, acrônimo para ***T****ransmuter*, ***R****educer*, ***OP****timizer*, ***E****xpander* e ***S****elector*, cinco esquemas básicos que traduzem alguns padrões recorrentes em programas Gamma:

Transmuter: aplica a mesma operação em todos os elementos do multiconjunto enquanto a condição de reação for verdadeira. Por exemplo, imaginemos um multiconjunto cujos elementos são formados por tuplas do tipo (x, y, s). Uma operação que corresponde ao padrão *transmuter* seria, por exemplo, uma reação que calcule $s = x + y$ para todas as tuplas que estão contidas neste multiconjunto.

Reducer: corresponde exatamente à Redução de Dados (*Data Reduction*) explicada anteriormente. Um padrão onde o tamanho do multiconjunto é reduzido. Por exemplo, para um multiconjunto formado pelas mesmas tuplas do tipo (x, y, s), uma reação que ao comparar um par de tuplas $(x1, y1, s1)$ e $(x2, y2, s2)$, substitua as duas tuplas por uma do tipo $(x1 + x2, y1 + y2, s1 + s2)$.

OPtimizer: tem por objetivo otimizar o multiconjunto de acordo com um critério específico, sem alterar a estrutura do multiconjunto. É parecido com o padrão *transmuter*, mas ao contrário deste, só transforma os elementos que atendem a um critério determinado além de não alterar a quantidade nem a estrutura dos elementos do multiconjunto. Tomando o mesmo exemplo do padrão *transmuter*, imaginemos uma reação que calcule $s = x + y$, para tuplas do tipo (x, y, s) mas se e somente se a condição $x > y$ for satisfeita.

Expander: correspondente à Expansão de Dados (*Data Expansion*) mencionada no segundo parágrafo desta seção. Consiste em decompor os elementos de um multiconjunto em elementos básicos. Um exemplo poderia ser construído por uma reação que decompõe cada tupla (x, y, s) em elementos x, y e s.

Selector: já o padrão *selector* remove elementos específicos de um multiconjunto, agindo como um filtro. Por exemplo, uma reação que elimina do multiconjunto toda e qualquer tupla (x, y, s) cujo valor de s seja maior que um dado valor N.

Em [8], é apresentada uma definição formal para cada padrão TROPES:

$$T(C, f) = x \longrightarrow f(x) \Longleftarrow C(x)$$

$$R(C, f) = x, y \longrightarrow f(x, y) \Longleftarrow C(x, y)$$

$$O(<, f_1, f_2, S) = x, y \longrightarrow f_1(x, y), f_2(x, y) \Longleftarrow (f_1(x, y), f_2(x, y)) < (x, y) \\ and\ S(x, y)\ and\ S(f_1(x, y), f_2(x, y)) \quad (2.4)$$

$$E(C, f_1, f_2) = x \longrightarrow (f_1(x), f_2(x)) \Longleftarrow C(x)$$

$$S_{i,j}(C) = x_1, ..., x_i \longrightarrow x_j, ..., xi \Longleftarrow C(x_1, ..., x_i) \\ (where\ 1 < j \leqslant (i + j))$$

2.3 Estado da Arte

Até agora vimos os principais conceitos e mecanismos que definem o modelo computacional Gamma e seu estilo de programação. Nas seções a seguir veremos os principais trabalhos que se basearam no Gamma e algumas importantes implementações deste modelo.

2.3.1 Extensões

Com o passar do tempo, foram sendo verificadas algumas deficiências do Gamma, dentre as quais podemos relacionar as seguintes como principais [13]:

- Inexistência de operadores que permitam combinação de programas;
- Dificuldade para estruturação de dados ou especificação de estratégias de controle; e
- Dificuldade de alcançar um bom nível de eficiência em qualquer implementação da linguagem, devido à inexistência de um modelo arquitetural que comporte a explosão combinatorial imposta pela sua semântica.

Tendo em vista estas deficiências encontradas, foram propostas algumas extensões ao Gamma original, das quais as principais serão explanadas nas seções seguintes.

2.3.1.1 Gamma Estruturada

A proposta de Gamma é fornecer um formalismo para especificação de programas com a menor quantidade de restrição possível. Dessa maneira, numa primeira análise, o fato de não fornecer dados estruturados nem estruturas de controle parece estar bastante adequado para a proposta de Gamma. Por outro lado, se inseriu a possibilidade de representar estruturas de dados possibilitando a verificação de tipos em tempo de compilação e consequentemente tornando a abstração dos dados mais explícita para o usuário final.

Em 1997, foi proposta a idéia de *Structured Gamma*, por FRADET e LE MÉTAYER [13]. Basicamente o Gamma Estruturado fornece um multiconjunto estruturado e consequentemente suporte a tipos.

Um multiconjunto estruturado fornece uma ordenação de endereçamento entre seus elementos, o que preserva o princípio da localidade, já que os dados são independentes, evitando que o multiconjunto tenha que ser manipulado por inteiro. Assim, se condições de reações forem satisfeitas por conjuntos diferentes de elementos do multiconjunto, reações podem ocorrer de forma independente e paralela. Tal estruturação de multiconjuntos funciona como uma espécie de vizinhança entre as moléculas de uma solução química. Em outras palavras, multiconjuntos estruturados são uma facilidade sintática usada para tornar a organização dos dados explícita, conforme [13].

Já o suporte a estruturação de dados existente nesta extensão ao Gamma original, fornece ao programador uma maneira de criar suas estruturas de dados de forma concisa através de uma gramática livre de contexto. Dessa forma é possível representar a maioria das estruturas de dados existentes. A utilização de tipos garante que a estrutura do multiconjunto esteja consistente com o tipo de dado representado, garantindo assim que a estrutura definida não se degenere [10].

2.3.1.2 Composição de Operadores

Outra importante extensão do Gamma introduz os conceitos de operadores sequenciais e paralelos. Em [14], são abordados os conceitos e propriedades de tais operadores.

HANKIN *et al.* [14] utilizam a notação $P_1 \circ P_2$ para expressar sequencialidade e $P_1 + P_2$ indicando paralelismo, onde P_1 e P_2 representam duas reações quaisquer. No primeiro caso, $P_1 \circ P_2$, o multiconjunto estável obtido pela execução de P_1 é passado como argumento para a reação P_2, indicando uma dependência na execução destas reações. Já a composição $P_1 + P_2$ indica a interdependência entre P_1 e P_2 uma vez que estas podem ocorrer em qualquer ordem, possivelmente em paralelo, e que a computação termina quando nenhuma das duas reações puderem mais reagir.

Outras composições semânticas da linguagem foram propostas, como por exemplo em [15, 16].

2.3.1.3 High Order Gamma

Como vimos anteriormente, a utilização de operadores de sequencialidade e paralelismo proporciona ao programador a possibilidade de desenvolvimento de melhores estratégias de controle para programas em Gamma. Dessa forma, uma outra abordagem para introdução de composição de operadores em uma linguagem consiste em fornecer ao programador uma maneira de defini-los como programas de ordem superior [12] (*Higher Order Programs*).Dessa maneira, a primeira extensão de ordem superior de Gamma foi proposta em 1994 por MÉTAYER [17].

Até agora, Gamma utilizava, basicamente, dois tipos de termos: multiconjuntos e programas. Por programa, entende-se um conjunto de regras de reescrita de multiconjuntos, ou seja, um programa consiste numa coleção de pares que envolvem condições de reação e ações a serem executadas. Já o multiconjunto é a base de dados única do modelo computacional em pauta. A proposta desta extensão de ordem superior é unificar estas duas categorias de expressões em uma única notação de configuração.

Outros trabalhos envolvendo extensões de ordem superior foram propostos como em [18–20].

2.3.2 Aplicações e Implementações

Como resultado do estudo e pesquisa envolvendo o paradigma Gamma, algumas implementações e aplicações foram propostas. Nesta seção abordaremos os principais trabalhos neste sentido. Em [9], encontram-se alguns exemplos de aplicações.

De acordo com [8, 12], na área de processamento de imagens, destacam-se duas aplicações, onde Gamma é utilizada. A primeira delas consiste numa aplicação para o reconhecimento da topografia tridimensional da rede vascular cerebral. Já a segunda diz respeito à geração de fractais para modelar o aumento de objetos biológicos. Também vale mencionar, o trabalho que define um Kernel de um Sistema Operacional em Gamma.

Com relação às implementações de Gamma, [12] divide estas em implementações para ambientes de memória distribuída e memória compartilhada. No primeiro caso, ambiente de memória distribuída, temos duas possíveis abordagens, que diferem na forma de controle, seja este centralizado ou distribuído.

No **controle centralizado** (memória distribuída), os elementos do multiconjunto estão distribuídos na memória local de cada processador e existe um controlador central, conectado a todos os processadores e responsável pela gerência de

troca de mensagens. Como exemplo de implementações deste caso podemos citar a *Conection Machine* [21], o *Maspar* [22], entre outras. A Connection Machine é um computador paralelo com $64K$ processadores. Em [21] os autores propõem um modelo síncrono de execução em Gamma e implementam tal modelo na *Connection Machine*.

Já no caso de ambiente de **controle distribuído** (em memória distribuída), toda a troca de mensagens entre os processadores ocorre de maneira assíncrona. Não existe a figura do controlador central e cada processador conhece somente seus processadores vizinhos. Como exemplo, pode-se citar a solução implementada na *Intel iPSC2 Machine* [23].

Temos ainda as implementações para ambientes de memória compartilhada. Neste caso, o modelo Gamma é visto como um modelo de memória compartilhada. Uma arquitetura de software específica foi proposta em [24].

Finalmente, serão listados abaixo alguns trabalhos importantes que não foram elencados nos tópicos acima por estarem classificados entre extensões do Gamma e implementações.

CHAM, um acrônimo para *CHemical Abstract Machine* foi proposta em 1992 por BERRY e BOUDOL [25]. Este trabalho foi especificamente importante pelos conceitos dos mecanismos de *membrane* e *airlock* trazidos. *Membranes* são utilizadas para encapsular soluções e forçar a ocorrência de reações localmente, ao passo que o mecanismo de *airlock* é utilizado para especificar as comunicações entre estas soluções encapsuladas e o ambiente.

HOCL (*High-Order Chemical Language*) foi proposto em 2009 por WANG e PRIOL [26]. Trata-se de uma linguagem baseada em uma extensão de alto nível do Gamma. Tem por objetivo fornecer uma maneira de definir regras para execução de reações e organização de elementos pelo ponto de vista HOCL de programação. Juntamente com o guia HOCL de programação, foi fornecido um compilador HOCL desenvolvido em JAVA.

Vale ressaltar que existe um trabalho do Professor Juarez Muylaert Filho que, apesar de caracterizar uma implementação do modelo computacional Gamma, não será abordado nesta seção. Isto se deve ao fato de tal implementação ter sido utilizada como modelo base para o problema sugerido por esta dissertação, por isso, veremos mais a frente no Capítulo 4 que descreve a solução proposta por este trabalho. A implementação do Professor Muylaert foi utilizada por PAILLARD [10], onde maiores informações também podem ser obtidas. Portanto, passaremos agora para os conhecimentos inerentes à fusão de dados.

Capítulo 3

Fusão de Dados para Acompanhamento de Contatos

Como abordado na introdução desta dissertação, o foco deste trabalho reside na implementação de um método de fusão de dados sobre o modelo computacional Gamma. Desta forma, é mister o entendimento do algoritmo de fusão de dados que foi utilizado como referência para a implementação em Gamma. Assim, uma vez terminada a abordagem dos assuntos inerentes ao modelo computacional Gamma, passaremos a abordar a fusão de dados aplicada ao acompanhamento de contatos.

3.1 Introdução

Conforme podemos verificar em [27], nos últimos anos, tem-se dispendido bastante esforço em pesquisa relacionada à fusão de dados, tanto para aplicações militares, quanto aplicações civis. No meio militar, mais especificamente no meio naval, tal assunto é foco de pesquisas e parte integrante de importantes projetos desenvolvidos pela Marinha, tanto no Brasil quanto em outros países.

Atualmente, uma das principais atribuições das forças navais é a patrulha de suas costas. No caso específico da Marinha do Brasil, tal missão é ainda mais importante devido a existência da Amazônia Azul [1]. Para cumprir tal finalidade, se faz necessário o processamento de dados provenientes de uma série de sensores distribuídos em pontos estratégicos e de grande tráfego. Estas informações podem ser adquiridas através de radares, sonares, sensores infravermelhos, boias radiosônicas, informações fornecidas por operadores advindas de observações visuais, entre outras. Sistemas de apoio à decisão e sistemas de vigilância, muitas das vezes integrados, são utilizados pelo operador que é responsável por coordenar este cenário. Dessa

[1] Área Oceânica, adjacente ao continente brasileiro, de importância estratégica para o Brasil e que corresponde a 52% da área territorial continental de nosso país.

maneira, seria bastante difícil a tomada de decisão em tempo hábil por parte do operador que controla este fluxo de embarcações, sem a utilização de métodos eficientes de fusão de dados.

Do ponto de vista tático, identificamos uma situação semelhante. Os meios navais (navios) pertencentes à Marinha possuem sistemas de comando, controle, comunicações e inteligência (C3I) [4], através dos quais é possível monitorar o cenário tático ao qual o navio se encontra inserido. Desta forma, o navio consegue identificar, monitorar e acompanhar possíveis alvos inimigos. Mais uma vez um método eficiente para a fusão de dados provenientes de múltiplos sensores torna-se extremamente importante do ponto de vista tático-operacional, sendo de importância estratégica para qualquer Marinha.

Por fim, vale mencionar que, para fins deste trabalho, os termos alvos ou contatos serão utilizados para referenciar objetos de interesse a serem acompanhados por um sistema específico.

3.1.1 O Conceito de Fusão de Dados

O conceito de fusão de dados foi utilizado e aplicado em diversas áreas de conhecimento e obteve grande divulgação no meio acadêmico, o que levou ao desenvolvimento de uma série de metodologias similares utilizadas para a resolução dos mesmos problemas.

Em 1985 foi desenvolvido um modelo de fusão de dados pela *U.S. Joint Directors of Laboratories, Data Fusion Group* (JDL) [27]. O objetivo do modelo JDL é fornecer uma conceituação e terminologias comuns através de um modelo de fusão de dados genérico e aplicável em muitas áreas de atuação. Tal modelo pode ser utilizado em diversos níveis de um sistema, como veremos nas seções seguintes.

Dessa forma, de acordo com o modelo JDL, podemos definir fusão de dados da seguinte maneira [28]:

> Um processo para lidar com associação, correlação e combinação de dados e informações de fontes únicas e múltiplas para alcançar uma posição refinada e estimativas de identidade, e avaliações completas e atualizadas de situações e ameaças, e seu significado. O processo é caracterizado pelo refinamento contínuo de suas estimativas e avaliações, e a avaliação da necessidade de fontes adicionais, ou alteração do próprio processo, para alcançar melhores resultados (WHITE e F.E. [28]).

Já em [4], baseado na definição anterior, encontramos uma definição um pouco mais concisa:

> Genericamente, pode-se dizer que o objetivo do processo de fusão é a combinação dos dados e conhecimentos de diferentes fontes, com o objetivo de maximizar o uso das informações, eliminando redundâncias, diminuindo a incerteza e aumentando a confiabilidade das informações (BARBOSA [4]).

Especificamente para o nosso contexto, aplicaremos o conceito de fusão de dados no sentido da utilização de um conjunto de informações, muitas vezes redundantes, de diversos sensores com o objetivo de conhecer o caminho percorrido por um alvo no cenário tático e com isso termos a condição de identificar em tempo real a localização atual de tal objeto de interesse.

3.1.2 Níveis de Fusão

Como o modelo JDL foi proposto como forma de equalização das terminologias e conceitos inerentes à fusão de dados, tal tarefa foi alcançada através da elaboração de um modelo funcional composto por vários níveis e que compreende qualquer sistema de fusão de dados existente. O modelo inicialmente proposto foi revisado e atualizado por HALL e MCMULLEN [29], conforme podemos verificar na Figura 3.1, que propõe as definições de níveis constantes da Tabela 3.1. O modelo consta de cinco níveis de fusão de dados onde o conceito pode ser aplicado desde o nível de sinais até informações mais refinadas que dizem respeito à informações de apoio aos objetivos da missão. Genericamente temos:

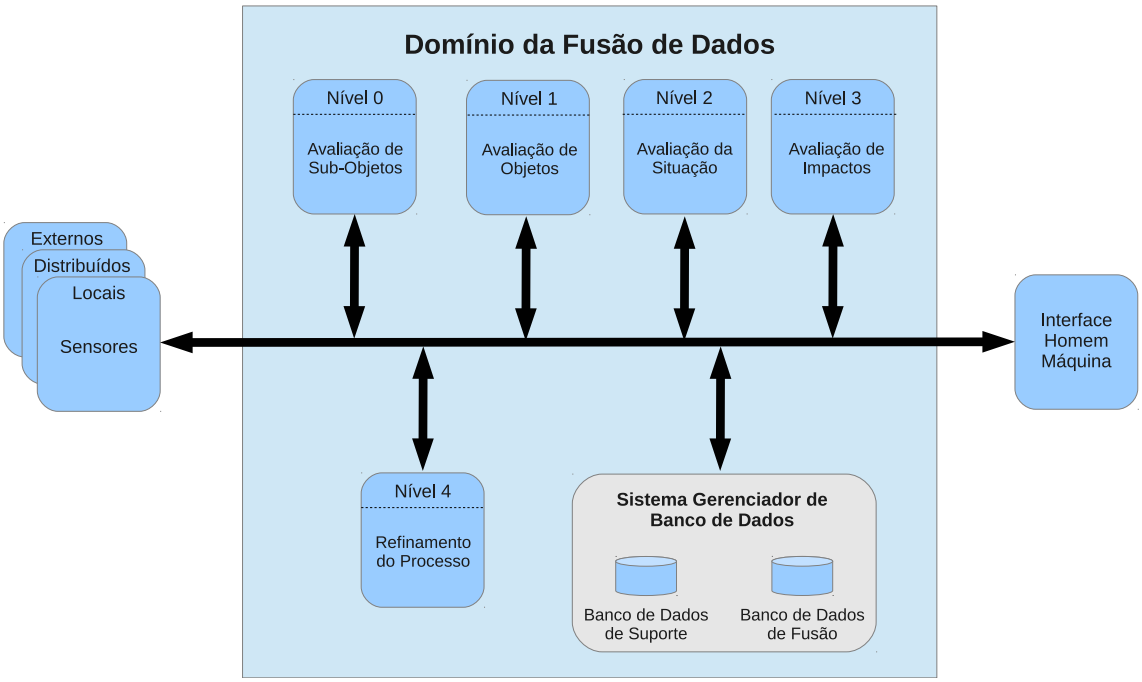

Figura 3.1: Modelo JDL revisado (Adaptada de [27]).

- Nível Zero: pode ser considerado um pré-processamento onde são obtidos os dados iniciais e características dos alvos observados;
- Nível um: neste nível ocorre uma combinação dos dados adquiridos pelos sensores, promovendo uma estimativa mais precisa da posição, velocidade e demais atributos das entidades envolvidas, dando suporte à predições futuras;
- Nível dois: ocorre a descrição dos relacionamentos correntes entre as entidades e eventos, no contexto do ambiente atual;
- Nível três: realização de projeção da situação, onde inferências sobre ameaças futuras e vulnerabilidades de forças amigas e inimigas são efetuadas; e
- Nível quatro: neste nível todo o processo de fusão de dados é monitorado com a finalidade de obter melhorias na performance do sistema em tempo real.

Tabela 3.1: Níveis de fusão

Nível de Fusão de Dados	**Descrição**
Nível 0 - Avaliação de Sub-Objetos	Estimação e Predição de estados observáveis de objetos baseados em caracterização e associação de dados a nível de sinais
Nível 1 - Avaliação de Objetos	Estimação e Predição de estados de entidades baseados em inferências a partir de observações
Nível 2 - Avaliação de Situação	Estimação e Predição de estados de entidades baseados em relações inferidas entre entidades
Nível 3 - Avaliação de Impactos	Estimação e Predição de efeitos sobre situações planejadas ou ações estimadas por parte dos participantes.
Nível 4 - Refinamento do Processo	Aquisição de dados adaptativa e processamento para apoiar objetivos da missão

3.1.3 A Abordagem Multisensor

Tratando-se de fusão de dados, uma abordagem multisensor apresenta vantagens comparada à abordagem de um único sensor. Em [27], são elencadas três principais vantagens:

- Em virtude da utilização de vários sensores idênticos (por exemplo, utilização de radares para o acompanhamento do movimento de um alvo), a combinação destes dados resultará em uma melhor estimativa de posição de destino e velocidade;

- Múltiplos sensores podem utilizar posicionamento relativo ou movimentar-se com a finalidade de aprimorar o processo de observação, como por exemplo, o processo de determinação de posicionamento de objetos através de triangulação; e

- A combinação de múltiplos sensores pode melhorar a capacidade de observação. O aumento dos pontos de observação pode trazer melhorias significativas para este processo. Tomemos como exemplo uma aeronave sendo observada por um radar e por um sensor de imagem infravermelho. O primeiro sensor pode determinar o alcance [2] da aeronave porém possui limitações no que diz respeito à determinação de sua orientação angular [3]. Já o sensor de imagem infravermelho pode determinar com precisão tal direção angular sendo ineficiente para a determinação de alcance. A correta associação destas duas observações pode levar a resultados melhores dos que poderiam ser alcançados pela utilização destes sensores separadamente.

Segundo KOCH [30] existem três arquiteturas aplicadas ao ambiente *multisensor*. A primeira delas discorre sobre a fusão direta dos dados do sensor. Neste caso, a fusão a nível de dados ocorre logo após os dados terem sido adquiridos e terem passado pelo processo de associação. Após esta fusão de dados, ocorre a extração das características para a futura declaração de identidade do objeto de interesse.

Um segundo modelo realiza a fusão de dados após estes terem sido adquiridos e terem suas características extraídas. Aqui ocorre inicialmente a representação dos dados em um vetor de características para a posterior fusão de dados e declaração de identidade, ao contrário da primeira abordagem que realiza a fusão logo apos as informações terem sido coletadas.

Por fim, a terceira abordagem propõe um processamento prévio sobre os dados de cada sensor produzindo inferências e decisões de alto nível que serão posteriormente combinadas através de fusão de dados.

[2] Neste caso o alcance diz respeito à coordenadas em um plano cartesiano de um determinado alvo adquiridas através de um radar.

[3] Orientação Angular diz respeito a informações de altitude da aeronave em questão.

3.1.4 Aplicações Militares e Não Militares

Técnicas e modelos de fusão de dados são largamente utilizadas atualmente. De acordo com [27], podemos encontrar a fusão de dados nas áreas acadêmica, comercial e industrial. Aplicações como controle automatizado de sistemas de gerenciamento industrial, robótica, construção civil, aplicações médicas, meteorologia, agricultura, aplicações que envolvem fenômenos naturais e prevenção de desastres, etc. Muitos destes utilizam sistemas de imagem por sensores multi espectrais. Técnicas normalmente aplicadas a fusão de imagens provenientes de multi sensores envolvem redes neurais adaptativas. Para estes tipos de aplicações ditas não militares, os dados podem ser adquiridos através de sensores específicos como acelerômetros, sinais acústicos, magnéticos, sensores de temperatura, etc. Numa visão mais ampla, podemos afirmar que estes dados são coletados de plataformas de sensores, como por exemplo satélites, aeronaves, navios, dados provenientes de observação humana, etc.

Da mesma forma, no ambiente militar, técnicas de fusão de dados são especialmente úteis e também muito utilizadas. HALL e LLINAS [27] mencionam que o Departamento de Defesa dos EUA foca o problema na localização, caracterização e identificação de entidades dinâmicas com interesse militar. Tais entidades podem corresponder a plataformas, armas e unidades militares. Como exemplo de aplicações de responsabilidade do Departamento de Defesa onde a fusão de dados é aplicada, podemos citar os sistemas de vigilância oceânica, defesa aérea, sistemas de inteligência no campo de batalha, vigilância e aquisição de alvos e, por fim, sistemas de alertas estratégicos e defesa. As informações que serão processadas por estes sistemas podem ser coletadas através de sonares, radares, sensores de raios infravermelhos, transdutores instalados nos veículos que fazem parte do cenário tático, dispositivos de comunicação que repassam informações sobre alvos e dados provenientes de observações humanas [4].

Especificamente no meio naval, as informações sobre objetos de interesse (alvos) são dispostos em mapas ou sistemas georreferenciados. BARBOSA [4] afirma que sobre tais alvos, diversas ações podem ser executadas, por exemplo, identificação de existência, determinação de identidade, extração de características cinemáticas para acompanhamento, determinação de posicionamento, classificação do grau de hostilidade, etc.

Portanto, para uma eficaz utilização do poder naval em um cenário tático-militar complexo, onde dados são recebidos por diversos sensores e onde o tempo de resposta do sistema está intimamente ligado à eficiente utilização de um meio militar, a utilização de técnicas de fusão de dados tornam-se estrategicamente importantes.

3.2 Acompanhamento (track)

3.2.1 Descrição

A necessidade de criação de algoritmos para rastreamento ou acompanhamento (*tracking algorithms*) surgiu com o desenvolvimento do radar, durante a segunda guerra mundial. Em meados de 1950, os radares já existiam a bordo de navios e aeronaves, porém o acompanhamento de alvos radar ainda era feito de maneira manual. Dessa forma, iniciaram-se estudos visando o desenvolvimento de algoritmos para acompanhamento de contatos. Conforme [27], o processo de acompanhamento de contatos está envolto em uma atmosfera de incerteza de medições, uma vez que, muitos ruídos e dados desnecessários são adquiridos pelos sensores. Assim, diante destas incertezas de medições, surgem dois obstáculos que devem ser levados em consideração por um algoritmo de acompanhamento de alvos.

O primeiro diz respeito a, mesmo em um ambiente de um único alvo (*Single Target*), como representar estas incertezas de medições. A segunda dificuldade é determinar como interpolar a real trajetória do alvo a partir de múltiplas medidas, uma vez que cada medição possui sua própria característica de erro associada.

Num ambiente com vários alvos, o problema ainda é maior, pois além de expressar a trajetória de cada alvo, dado o ambiente de incertezas, diversas trajetórias podem sofrer interseção o que insere mais um fator complicador ao algoritmo de acompanhamento, como exemplificado na Figura 3.2. Nesta figura os pontos em azul correspondem à dados adquiridos por sensores e os dados que não foram associados através de setas quando $T=5$, correspondem a dados não válidos (por exemplo, ruídos). O problema é considerado ainda maior no caso de um ambiente em três dimensões, como por exemplo envolvendo alvos marítimos, terrestres e aéreos.

Dada uma aplicação para acompanhamento de contatos, definimos por "acompanhamentos"(*tracks*) aqueles objetos móveis que estão sendo observados e monitorados por tal aplicação através de processos de medições com o objetivo de manter uma estimativa de seu estado corrente [4]. Tal estado corrente é composto principalmente por informações cinemáticas (posicionamento, velocidade, aceleração, etc.) e informações sobre as características físicas do sensor.

Por fim, uma vez estipulados os "acompanhamentos", a partir de sua posição, podem ser estimados seu rumo, velocidade e posicionamento futuro.

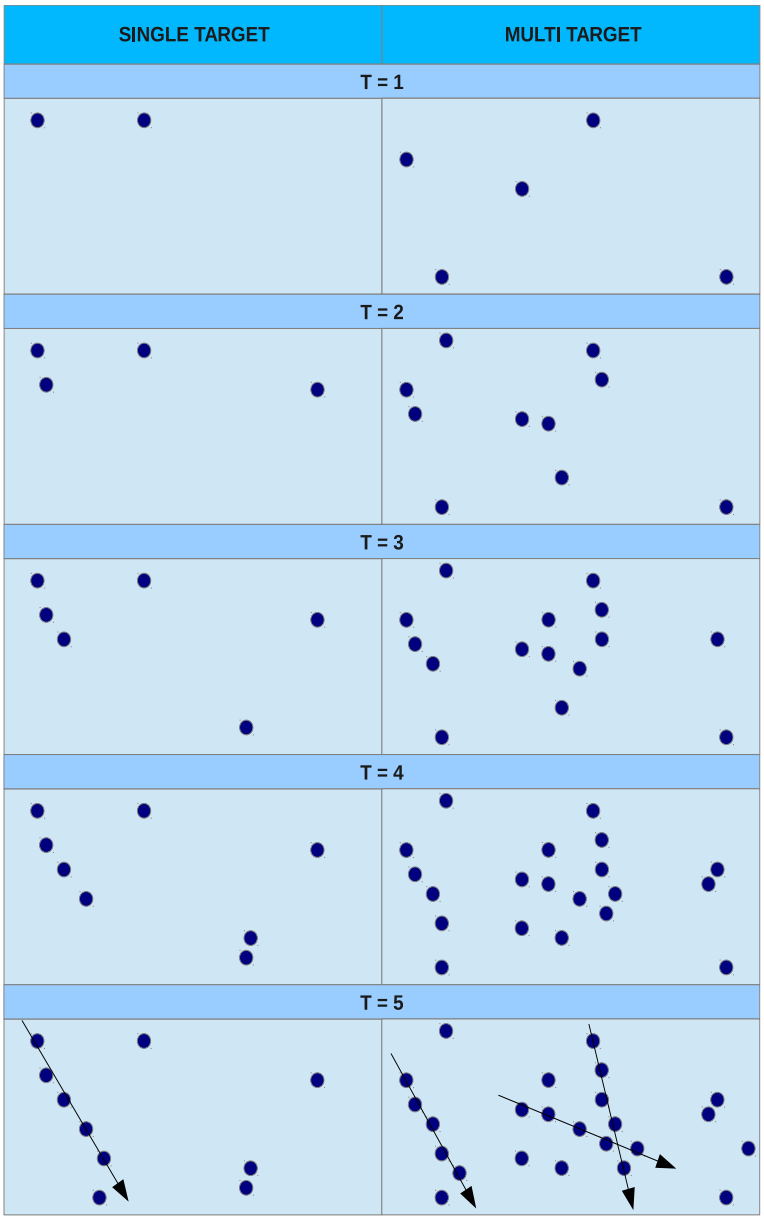

Figura 3.2: Determinação de trajetórias de alvos

3.2.2 Associação e Estimativa

Relacionados ao processo de fusão de dados, podemos elencar dois importantes conceitos: a associação e estimativa.

Segundo [31], podemos definir estimativa como sendo o processo de inferir indiretamente o valor de uma quantidade de interesse, a partir de observações incertas e imprecisas. Este processo também tem por objetivo minimizar os erros de medição, inerentes ao sensores e os erros de processos, inerentes à fatores aleatórios tais como condições ambientais.

Para permitir a estimação, é necessária a correta associação das informações provenientes dos diversos sensores aos acompanhamentos que foram criados. Existem diversos métodos empregados para esta finalidade. Em [27], são elencados os

seguintes métodos: associação pelo vizinho mais próximo (*Nearest Neighbors*), associação por divisão de acompanhamentos (*Track Splitting*), associação por múltiplas hipóteses (*Multiple Hypothesis*), associação por janelas (*Gating*), busca binária (*Binary Search*) e associação por árvores (*kd-trees*). Além desses métodos, BARBOSA [4], cita métodos probabilísticos como associação probabilística de dados (*Probabilistic Data Association*), associação probabilística conjunta de dados (*Joint Probabilistic Data Association*), entre outros.

Basicamente, podemos citar que em um sistema de fusão de dados, os elementos de interesse são identificados e passam a ser encarados como "acompanhamentos". Uma vez criados os acompanhamentos, estes necessitarão ser associados às medidas que serão recebidas pelos sensores, o que visa permitir a atualização das informações referentes à cinemática do objeto de interesse em questão. A partir das associações elaboradas, são realizadas predições com estimativas do posicionamento do alvo.

Nas próximas seções veremos maiores detalhes sobre o método de associação baseado em múltiplas hipóteses. Este é o método clássico e mais empregado em ambientes que utilizam múltiplos sensores para múltiplos alvos [32].

3.2.3 O MHT - Multiple Hypothesis Tracking

O *Multiple Hypothesis Tracking - MHT*, é considerado o método clássico e mais utilizado para associação de contatos em um sistema de acompanhamento de múltiplos alvos (*Multiple Target Tracking - MTT*) [4].

Como vimos anteriormente, o problema da fusão de dados envolve estimativa e associação de contatos. Porém, esta associação torna-se complicada num cenário onde existam múltiplos alvos a serem acompanhados, como por exemplo num sistema MTT.

No MHT, múltiplas hipóteses quanto ao posicionamento de determinado contato são geradas a partir das predições estabelecidas. Usualmente, utiliza-se uma janela em torno da predição de posicionamento do alvo com a finalidade de localizar possíveis candidatos para o posicionamento futuro do mesmo, dentre os pontos de medida localizados nesta janela, conforme ilustrado na Figura 3.3. Nesta figura, existe um conjunto de observações $(O_1, O_2, ..., O_{13})$ e uma janela relacionada a predição P_1, cujos candidatos a posicionamento futuro são O_7, O_8 e O_9.

Um acompanhamento é criado a partir de observações e as hipóteses são elaboradas a partir dos acompanhamentos existentes e das novas observações recebidas no instante de tempo subsequente. Assim, a cada novo instante de tempo, uma hipótese é expandida em um conjunto de novas hipóteses. É importante ressaltar que hipóteses são formadas por acompanhamentos compatíveis, ou seja, aqueles cujas observações não são comuns a nenhuma outra hipótese.

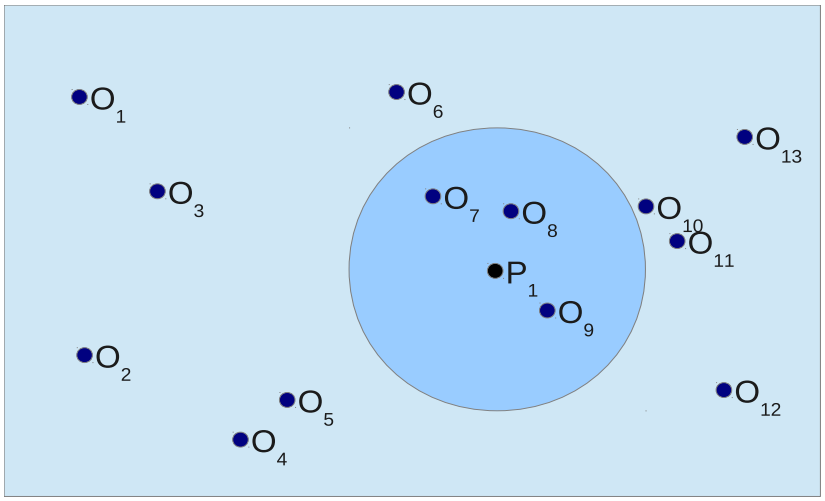

Figura 3.3: Janela de predição

Não somente as hipóteses, mas também os acompanhamentos são atualizados mediante à recepção de novas medições, a cada instante de tempo. A recepção destas novas informações tem por finalidade atualizar tais acompanhamentos. Por exemplo, no caso de medições adquiridas por um radar, a cada instante de tempo novas medições são recebidas pelo sistema que implementa a fusão de dados. Expressões baseadas em probabilidades são responsáveis por estas alterações, utilizando como parâmetros de probabilidade da presença prévia do alvo, a consistência cinemática das observações contidas em um acompanhamento, por exemplo.

A cada acompanhamento pertencente a uma hipótese são atribuídos valores de avaliação, conforme a persistência deste objeto durante certo número de atualizações (razão de verossimilhança [33]). Assim o valor da probabilidade correspondente a um acompanhamento é dado pela soma das probabilidades de todas as hipóteses que possuem este acompanhamento.

Por fim, vale mencionar que existem diversas técnicas de exclusão e aglutinação de acompanhamentos e hipóteses, como por exemplo a clusterização [4], com a finalidade de redução do volume de dados a serem processados, o que permite ao MHT ser utilizado em cenários complexos com grande número de alvos e ruídos, como por exemplo um cenário tático naval onde os navios de uma Marinha de guerra encontram-se inseridos.

3.3 Fusão de Múltiplos Sensores através da utilização de Grafos

A seguir veremos a utilização de grafos no contexto da fusão de dados, abordando os principais métodos de fusão de dados para acompanhamento de contatos cuja solução seja baseada em grafos.

3.3.1 Métodos Baseados em Grafos

Uma das maneiras de representar os dados recebidos dos sensores objetivando a escolha da melhor hipótese de caminho de um contato é através da utilização de grafos. Diversos autores utilizam esta abordagem, como por exemplo [27, 34, 35]. O objetivo da utilização de grafos para tal finalidade é a possibilidade de utilização de algoritmos de otimização de caminhos, visando a escolha da trajetória mais provável de determinado alvo. Ou seja, a partir de um grafo cujas arestas possuem custos associados à informações cinemáticas de um alvo em questão, um algoritmo de otimização escolhe a melhor hipótese de caminho (caminho mínimo) para determinado alvo. Para a realização desta otimização utiliza-se algoritmos de caminho mínimo, onde podemos citar como principais os algoritmos de Dijkstra, Ford/Moore, Floyd e Dantzig [4].

Nos grafos utilizados, as arestas possuem custos associados. O caminho mínimo será aquele cuja soma dos custos associados as todas as arestas pertencentes a um determinado caminho for a menor. Com relação aos vértices, duas abordagens podem ser utilizadas. Na primeira cada vértice representa uma posição do alvo recebida pelo sensor ou um ruído associado. Numa segunda abordagem, os vértices do grafo representam estados estimados das medidas fornecidas. Para escopo deste trabalho, o algoritmo de referência baseia-se na primeira abordagem.

3.3.2 PP - Pares de Plots

Cada informação adquirida dos sensores que representam possíveis detecções de alvos ou falsos alarmes são conhecidas pelo termo *Plot*. A maioria dos métodos existentes utilizavam plots como vértices dos grafos, o que significa que os dados (plots) representam pontos no plano cartesiano e, portanto, nenhuma informação sobre aceleração era levada em consideração pelos algoritmos de otimização.

Assim, LIVERNET e CUILLIERE [5] propõem uma nova metodologia para a utilização de grafos associados a fusão de dados para acompanhamento de contatos. Os autores utilizam um grafo de par de plots, ao invés de um grafo de plot.

Num grafo de par de plots os vértices correspondem às arestas de um grafo de plot, passando do domínio do posicionamento para o domínio das velocidades. Em

outras palavras, o grafo de par de plots é o *line graph* [4] do grafo de plots.

Os passos da construção de um grafo de par de plots podem ser resumidos da seguinte maneira:

- Inicialmente são criados grafos com os plots recebidos por cada sensor (grafo de plots). É importante lembrar que os critérios para a criação das arestas neste grafo levam em consideração a consistência cinemática do alvo em questão, que está ligada ao deslocamento e velocidade máximas admissíveis para o alvo. Assim, conforme [4], a decisão de criação de uma aresta entre dois plots deve respeitar $D < V_{max} \triangle t + f(\varepsilon)$, onde:

  D - Distância entre dois plots;

  $\triangle t$ - Tempo decorrido entre o recebimento de dois plots;

  V_{max} - Velocidade máxima do contato; e

  $f(\varepsilon)$ - Incerteza na distância.

- Com o grafo de plots formado para cada sensor, cada aresta passa a ser um vértice do grafo de par de plots, ou seja, os vértices de um "grafo de par de plots"são formados pela utilização de dois plots de um "grafo de plot".

Na Figura 3.4 podemos verificar os passos para a criação de um grafo de par de plots a partir de medidas provenientes de dois sensores, segundo a abordagem original [5]. Os losangos em preto correspondem a plots do sensor de número 1, enquanto que os quadrados vermelhos indicam medições pertencentes ao sensor número 2. Nos grafos de plots de cada sensor, as arestas possuem triângulos e círculos, correspondendo a vértices nos grafos de par de plots dos sensores 1 e 2, respectivamente. Ou seja, para a criação do grafo de par de plots, cada aresta do grafo de plot será um vértice do grafo de par de plot. Dessa maneira, conforme mencionamos anteriormente, o grafo de par de plots é o grafo linha (*line graph*) do grafo de plots. Após a criação dos grafos de pares de plots dos dois sensores em questão, os mesmos são conectados por arestas adicionais, que preservam a mesma consistência cinemática citada anteriormente, criando um único grafo de par de plots que somente então será processado por um algoritmo de otimização visando a escolha da melhor hipótese de caminho percorrido pelo alvo. Por fim, são escolhidos os plots iniciais que pertencem ao melhor caminho escolhido no gráfico de par de plots com informações dos dois sensores.

[4]Grafo linha (line graph) é denotado por L(G) e representa a adjacência entre as arestas do grafo G. Assim, cada vértice de L(G) representa uma aresta de G. Dois vértices de L(G) são adjacentes se e somente se suas arestas correspondentes compartilham um mesmo vértice em G.

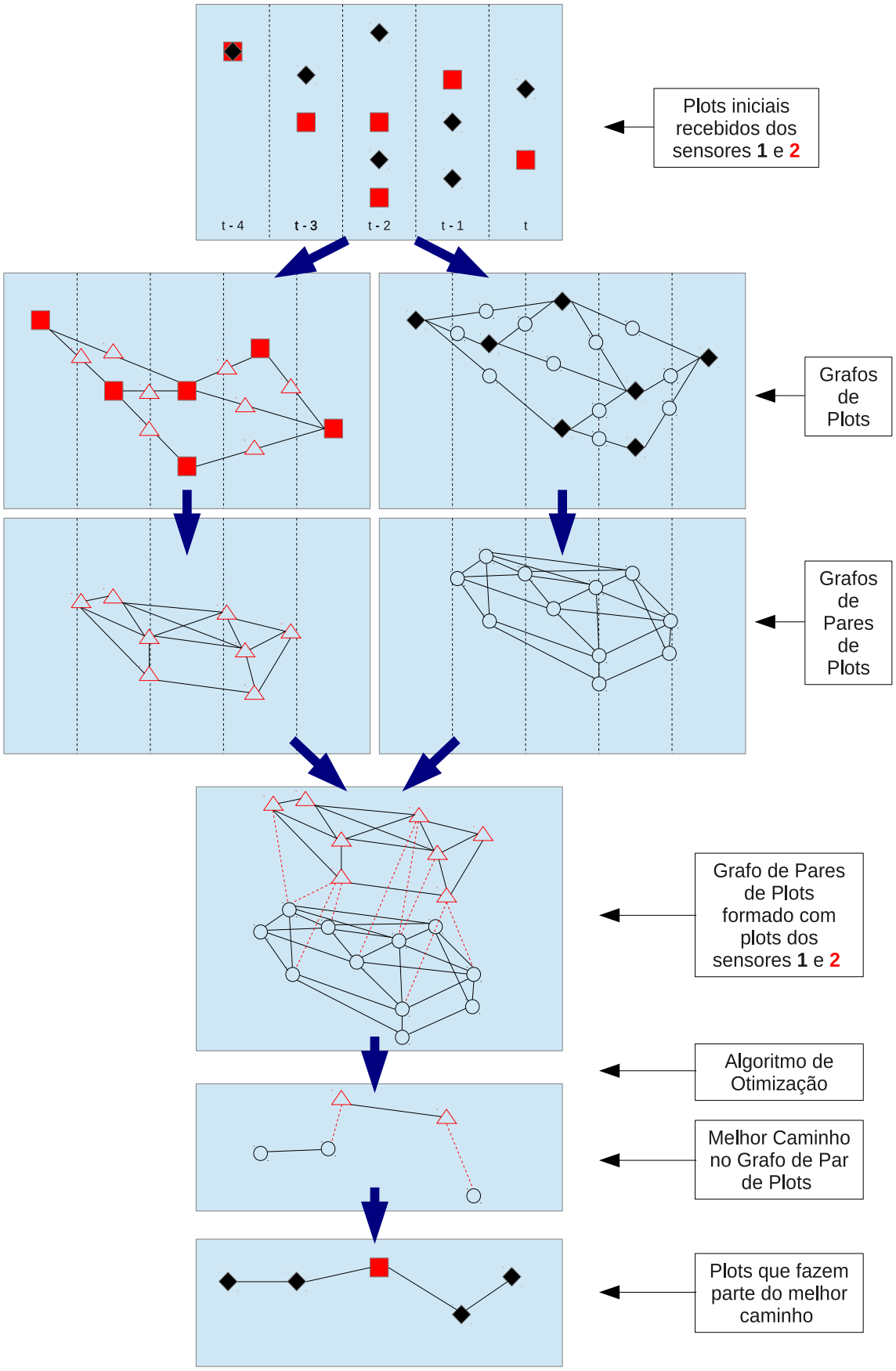

Figura 3.4: Formação do grafo de par de plots (Baseada em [4].)

3.3.3 PPDE - Pares de Plots em Dois Estágios

O modelo original ao utilizar a metodologia de pares de plots [5] aplica o algoritmo de Dantzig sobre o grafo de par de plots como algoritmo de otimização visando a escolha do caminho mínimo, ou seja, no exemplo da Figura 3.4, após o último grafo apresentado, o algoritmo de Dantzig é aplicado. Pela quantidade de hipóteses criadas pelo grafo de par de plots (que, conforme [4] cresce exponencialmente com o aumento do número de sensores ou quantidade de ruídos), o modelo de LIVERNET e CUILLIERE [5] utiliza uma janela de tempo reduzida, o que diminui a eficiência do algoritmo na escolha do melhor caminho. Além disso, o algoritmo de otimização é aplicado uma única vez sobre todos os pontos oriundos de todos os sensores o que, além de ser muito custoso do ponto de vista computacional, não viabiliza a paralelização na computação dos dados oriundos de cada sensor, uma vez que utiliza-se um único grafo cujos vértices correspondem a dados de todos os sensores envolvidos, diferentemente da proposta do PPDE, conforme veremos a seguir.

Diante deste cenário, em [4, 36] foi proposta a metodologia de grafos de pares de plots em dois estágios. Pela proposta, o processo de escolha do melhor caminho entre os grafos de pares de plots pode ser executada em paralelo. Além disso, a construção do grafo de par de plots se dá em dois estágios conforme veremos.

Primeiro Estágio

No primeiro estágio, as medições são recebidas dos sensores e, para o conjunto de observações de cada sensor, um grafo de plots é formado. Da mesma maneira que ocorre no método original [5], a condição para a criação das arestas no grafo de plots segue critérios de consistência cinemática. Com os grafos de plots formados para cada sensor, criam-se grafos de pares de plots, também por sensor. Agora reside uma das diferenças comparado ao método original: aplica-se o algoritmo de otimização (Dantzig) aos grafos de pares de plots de cada sensor, visando a escolha do melhor caminho por sensor, ao invés de criarmos um grafo de par de plots pela "conexão" dos grafos de pares de plots de todos os sensores, como realizado no método original. Neste ponto, o primeiro estágio é encerrado (Figura 3.5).

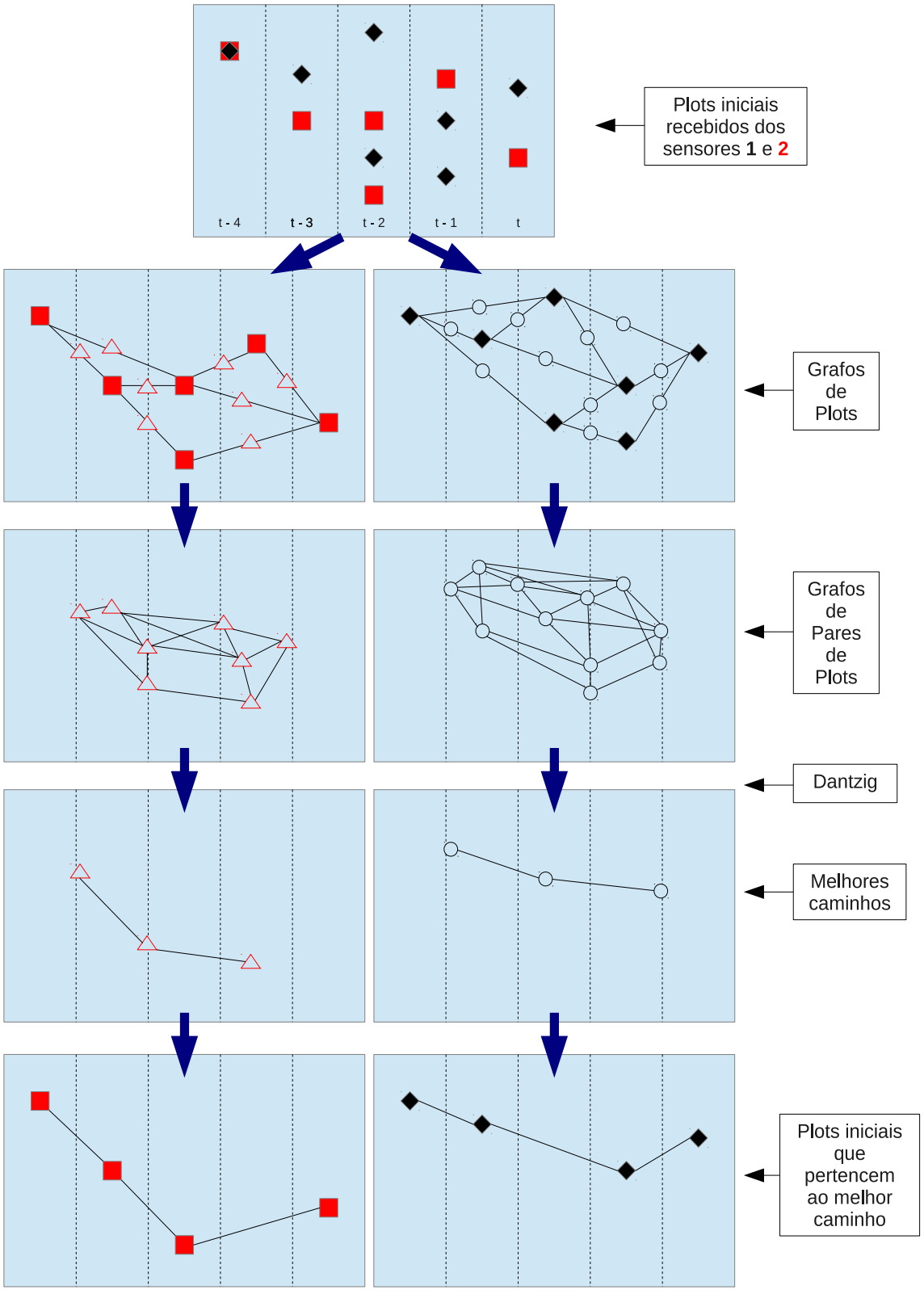

Figura 3.5: PPDE - Primeiro estágio

Segundo estágio

Por ocasião do início do segundo estágio, temos um conjunto de plots, de cada sensor, que pertencem ao melhor caminho calculado no primeiro estágio. Somente agora os dados de todos os sensores são combinados visando a criação de um novo grafo de plots e, posteriormente, seu correspondente grafo de par de plots. Agora, pela aplicação do algoritmo de Dantzig novamente, tem-se o melhor caminho do grafo.

As Figuras 3.5 e 3.6 ilustram a execução do primeiro e segundo estágio, respectivamente.

Vale mencionar que para a execução do algoritmo de Par de Plots em dois estágios se faz necessário o fornecimento de três plots iniciais que correspondam ao posicionamento real do alvo em questão. Estas três medidas serão utilizadas na inicialização do algoritmo para fornecer as características cinemáticas do alvo a ser acompanhado. Maiores informações sobre os três pontos iniciais mencionados serão vistas na Seção 4.2.

Por fim, a abordagem de grafos de pares de plots em dois estágios (PPDE), obteve ganhos de acurácia, permitiu a utilização de uma janela de tempo maior que a abordagem original, além de possibilitar o paralelismo do processamento do primeiro estágio, o que é ainda mais evidente no caso do aumento do número de sensores. A comparação entre as duas abordagens foi realizada em [5].

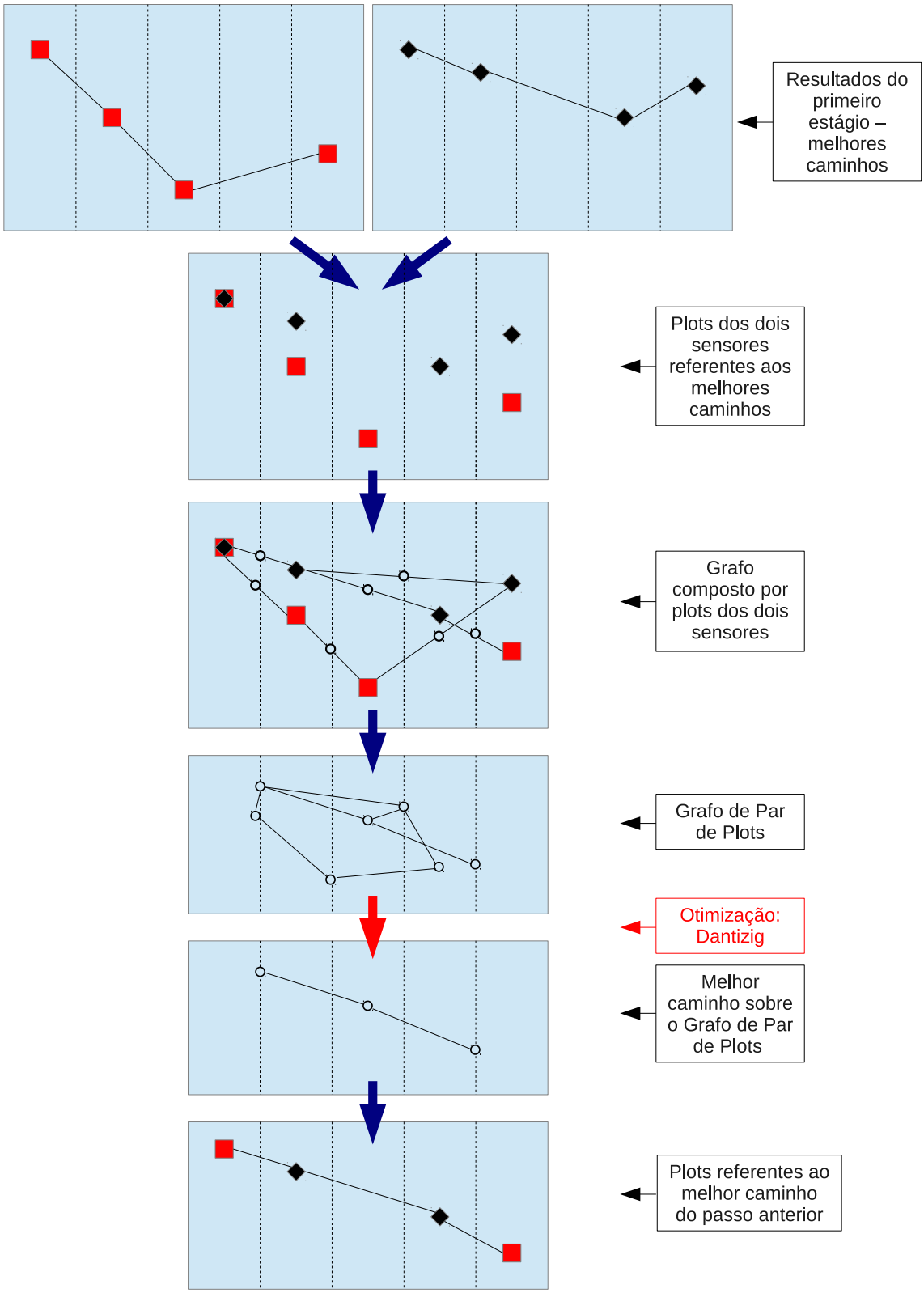

Figura 3.6: PPDE - Segundo estágio

Capítulo 4

Solução Proposta

Tendo finalizado a revisão da literatura, onde foram abordados os principais conceitos inerentes ao formalismo Gamma e à fusão de dados para acompanhamento de contatos, partiremos ao capítulo dedicado à solução proposta. Inicialmente, abordaremos em linhas gerais uma implementação real de Gamma, desenvolvida por Juarez Muylaert e Simon Gay. Tal implementação foi utilizada para o desenvolvimento de um algoritmo de fusão de dados, em Gamma, que será apresentado na Seção 4.3.

4.1 Implementações de Gamma Utilizadas

4.1.1 Descrição

Gamma foi incialmente proposto por BANÂTRE e LE MÉTAYER [3] em 1986 como um modelo computacional teórico, um formalismo para a especificação de programas segundo a metáfora de reações químicas e baseado na reescrita paralela de multiconjuntos, como vimos no Capítulo 2. Esta proposta inicial de Gamma levou a várias tentativas de implementação. Para fins deste trabalho, iremos nos basear em três implementações deste modelo, conforme listado a seguir:

- *Gamma-Sequencial*;
- *Gamma-MPI*; e
- *Gamma-GPU*.

As duas primeiras implementações foram desenvolvidas por Juarez Muylaert e Simon Gay. A implementação que denominamos *Gamma-Sequencial* utiliza somente um processador para a execução das reações e não permite a execução em hardware paralelo. Neste caso, a sensação de paralelismo é obtida através da eventual alternância entre a execução das reações em um único processador. Ou seja, conforme vimos anteriormente, o formalismo Gamma possui um paralelismo inerente

que corresponde à livre execução de reações em um multiconjunto. Assim, nesta implementação, tal característica de paralelismo é simulada através da alternância entre as execuções das reações envolvidas. Esta abordagem sequencial foi expandida pelos mesmos autores para a execução em um hardware paralelo onde a comunicação entre os diversos processadores ocorre através da utilização de uma interface de troca de mensagens, mais especificamente a *Message Passing Interface* (MPI) [6]. Nesta implementação, chamada de *Gamma-MPI*, as reações são executadas em processadores distintos de um ambiente de execução paralela, o que envolve o envio do multiconjunto para diversos processadores. A gerência do envio do multiconjunto e da execução das reações nos diversos processadores é realizada através de um conjunto de "células" fornecidas pela implementação. Vale mencionar que *Gamma-MPI* foi estendido por Gabriel Antoine Louis Paillard em 1999 [10, 37] com enfoque na criação e utilização de tipos de dados o que caracteriza esta extensão como uma implementação de Gamma Estruturada.

Já a terceira implementação, chamada de *Gamma-GPU* foi implementada por Rubens Pailo em 2015 [7] e consiste na extensão de *Gamma-MPI* para a execução em um hardware paralelo com suporte à GPU. Veremos a seguir as principais características comuns às três implementações mencionadas.

4.1.2 Principais Características

Para possibilitar a geração de um código executável a partir de um código estruturado de acordo com o formalismo Gamma [1], as três implementações verificadas na Subseção 4.1.1, utilizaram-se de um processo de compilação, escrito em linguagem C, cujas características serão descritas a seguir.

As tarefas iniciais de um procedimento de compilação envolvem a análise léxica e sintática. A primeira se encarrega de dividir a entrada em unidades discretas, chamadas de lexemas ou tokens, para a posterior verificação de relacionamento entre eles. Tais lexemas ou tokens correspondem a palavras reservadas, nomes de variáveis, constantes, operadores, símbolos especiais, entre outros. Já a fase de análise sintática é responsável por verificar a validade dos relacionamentos entre os lexemas através da aplicação das regras de uma gramática estabelecida. Por gramática conceituamos como sendo o conjunto de regras que definem os relacionamentos possíveis entre os lexemas de uma linguagem. Para o compilador desenvolvido, as fases de análise léxica e análise sintática foram desempenhadas pelas ferramentas *lex & yacc* da AT&T [38].

Após o cumprimento das fases anteriores a árvore gramatical está completa e,

[1] Para o escopo deste trabalho, todos os códigos escritos em linguagens de programação derivadas do formalismo Gamma para as implementações *Gamma-Sequencial*, *Gamma-MPI* e *Gamma-GPU* possuem a extensão *.gm*

com isso, uma série de ponteiros que referenciam diversas estruturas de dados relacionados à informações do programa de entrada estão prontos para gerar código fonte em linguagem C. Este código irá conter todas as rotinas necessárias à execução do código original em Gamma.

O código em linguagem C fornecido pelo compilador Gamma ainda passará por outro processamento com a finalidade de tornar-se código executável para determinada arquitetura de hardware através da utilização de um compilador específico. Para isso, o mesmo deverá ser compilado juntamente com outros arquivos que fazem parte do *framework* da implementação utilizada. No caso de *Gamma-Sequencial* utilizamos o compilador *gcc* [2] para compilar o arquivo "*.c*" gerado pelo compilador Gamma visando a obtenção de um código executável. Para a implementação *Gamma-MPI* utilizamos o compilador *mpicc*, ao passo que no caso de *Gamma-GPU* o compilador utilizado foi o *nvcc* [3].

Por fim, após o código Gamma ter passado por todas as etapas descritas acima, o mesmo será executado em alguma arquitetura de hardware específica, como por exemplo, em um processador com um único núcleo de processamento, no caso da utilização da implementação *Gamma-Sequencial*, um processador multicore ou em um cluster, no caso de *Gamma-MPI* ou ainda em um ambiente com suporte a GPU, no caso de *Gamma-GPU*.

Na Figura 4.1 temos um resumo das etapas acima mencionadas.

4.1.3 Sintaxe Suportada

Para expressar um programa em Gamma, se faz necessária a utilização de uma linguagem e sua respectiva sintaxe. Tendo em vista a simplicidade na expressão de um programa como sendo uma proposta do paradigma Gamma, é de se esperar que uma implementação real deste modelo forneça uma quantidade pequena de possibilidades no que diz respeito aos recursos para expressar um programa.

Dessa maneira, como iremos verificar a seguir, as implementações de Gamma utilizadas, sejam elas, *Gamma-Sequencial*, *Gamma-MPI* e *Gamma-GPU* utilizam o mesmo conjunto reduzido de possibilidades de recursos para definição de programas.

O conjunto de lexemas suportados nas implementações utilizadas neste trabalho abordam palavras reservadas, operadores lógicos, operadores relacionais, operadores aritméticos, operadores de composição e um conjunto de símbolos básicos.

[2] *GNU Compiler Collection*
[3] Acrônimo para *NVIDIA's CUDA Compiler*

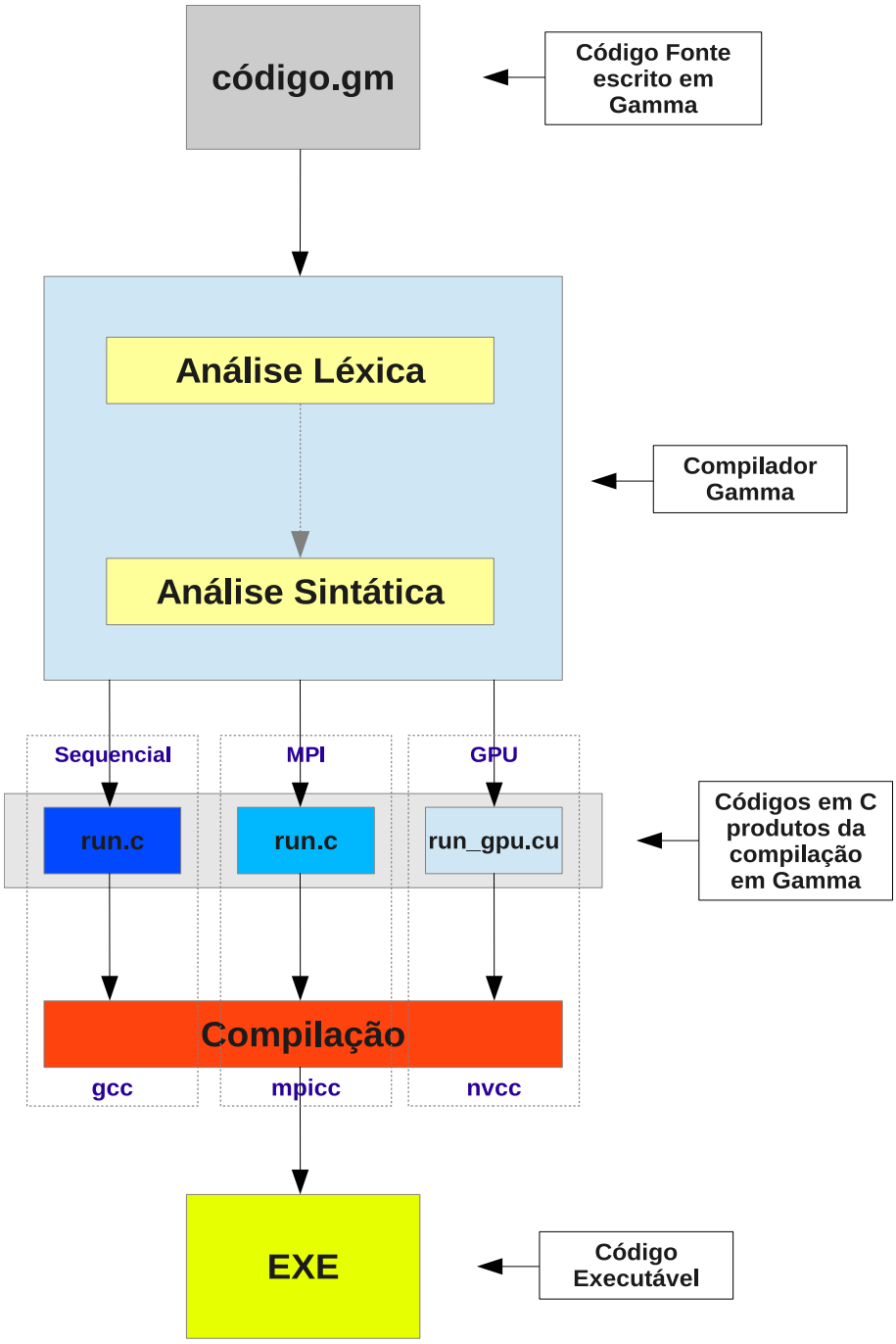

Figura 4.1: Processo de compilação nas implementações de Gamma utilizadas

Uma diferença significativa quanto a proposta original de Gamma consiste no fato de que as três implementações de Gamma nas quais estamos nos baseando suportam operadores de composição, como vimos no Capítulo 2. Tais operadores possibilitam a execução de reações em sequência ou em paralelo, através da utilização dos símbolos ";" e "|", respectivamente.

Como exemplo de um programa expresso na linguagem Gamma, segundo a implementação de Juarez Muylaert e Simon Gay, tomemos o exemplo do Capítulo 2, Equação 2.2, que implementa o problema da busca do menor elemento dado um

multiconjunto qualquer:

$$\begin{aligned} Menor(M) = \Gamma(R, A)\ \ (M)\ \ where \\ R(x, y) = x \geq y \\ A(x, y) = y \end{aligned} \tag{4.1}$$

O mesmo problema, expresso em Gamma, segundo à sintaxe fornecida pela implementação do Prof. Muylaert poderia ser expresso da seguinte forma:

$$\begin{aligned} Menor\ \ \{1, 2, 3, ..., N\} \\ where \\ Menor = replace\ \ x \\ by\ \ y \\ if\ \ x \geq y \end{aligned} \tag{4.2}$$

Dessa forma, genericamente temos:

$$\begin{aligned} R_1 \diamond R_2 \diamond R_3 \diamond R_4\ \ ...\ \ R_n\ \ \{x_1, x_2, x_3,\ \ ...\ \ , x_n\} \\ where \\ R_{1,2,3,\ \ ...\ \ ,n} = replace\ \ x_1, x_2, x_3,\ \ ...\ \ , x_n \\ by\ \ y_1, y_2, y_3,\ \ ...\ \ , y_n \\ if\ \ Condicao \end{aligned} \tag{4.3}$$

Onde:

- O símbolo $\diamond$ refere-se ao operador de composição que pode ser expressado por ";" ou "|", indicando execução de reações em sequência ou em paralelo, respectivamente;
- $\{x_1, x_2, x_3,\ \ ...\ \ , x_n\}$ expressa o multiconjunto em questão;
- x_n e y_n são elementos do multiconjunto que podem ser valores numéricos e/ou tuplas de tamanho N; e
- "*Condicao*" expressa uma condição lógica que deve ser satisfeita para que a operação de "substituir por"(*replace by*) ocorra.

Na Tabela 4.1 temos uma listagem dos símbolos que compõem a sintaxe suportada nas implementações de Gamma utilizadas.

Tabela 4.1: Símbolos suportados pelas implementações utilizadas (Baseado em [7]).

Palavra Reservada	
if	Teste da condição de reação
replace	Padrão de seleção de elementos do Multiconjunto
by	Ação (Reescrita) no Multiconjunto
where	Definição de reações
true	Valor booleano verdadeiro
false	Valor booleano falso
empty	Valor vazio
Operador Lógico	
and	Operação Lógica E
or	Operação Lógica OU
Operador Relacional	
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
==	Igual a
!=	Diferente de
Operador Aritmético	
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo
Operador de Composição	
;	Execução Sequencial
|	Execução Paralela
Símbolos	
,	Separador de Elementos
=	Atribuição
(	Início de Expressão
)	Fim de Expressão
[	Início de Tupla
]	Fim de Tupla
{	Início do Multiconjunto
}	Fim do Multiconjunto

4.2 Considerações sobre PP e PPDE

Antes de iniciarmos a descrição mais aprofundada dos detalhes relativos a solução desenvolvida, se faz necessária uma breve abordagem de alguns aspectos mais específicos sobre métodos de fusão de dados baseados em grafos, especialmente sobre o método de Pares de Plots em Dois Estágios (PPDE), que não foram abordados nos capítulos iniciais.

O PPDE foi inspirado no método de Pares de Plots (PP) e está implementado em um sistema de navegação desenvolvido pela Marinha do Brasil, chamado de Centro de Integração de Sensores e Navegação Eletrônica (CISNE). Este método é extremamente útil para a fusão de dados de múltiplos radares, especialmente em ambientes com taxas de ruídos elevadas.

Em radares, os dados são adquiridos mediante a varredura de uma área de amostragem que define o posicionamento e o tempo da marcação de cada plot (tempo de aparecimento). Vale a pena ressaltar que, conforme visto anteriormente, um plot pode significar o posicionamento real de algum contato ou um ruído de qualquer tipo. Para fins de implementação do PPDE, os plots sofrem um processo de aproximação de sua marcação de tempo onde este é decorrido na escala de centésimos. Isto significa que todos os plots recebem a marcação de tempo do centésimo mais próximo de sua marcação original, conforme ilustrado na Figura 4.2. Segundo informações dos autores do PPDE, este procedimento de aproximação visa facilitar a futura manipulação dos grafos utilizados.

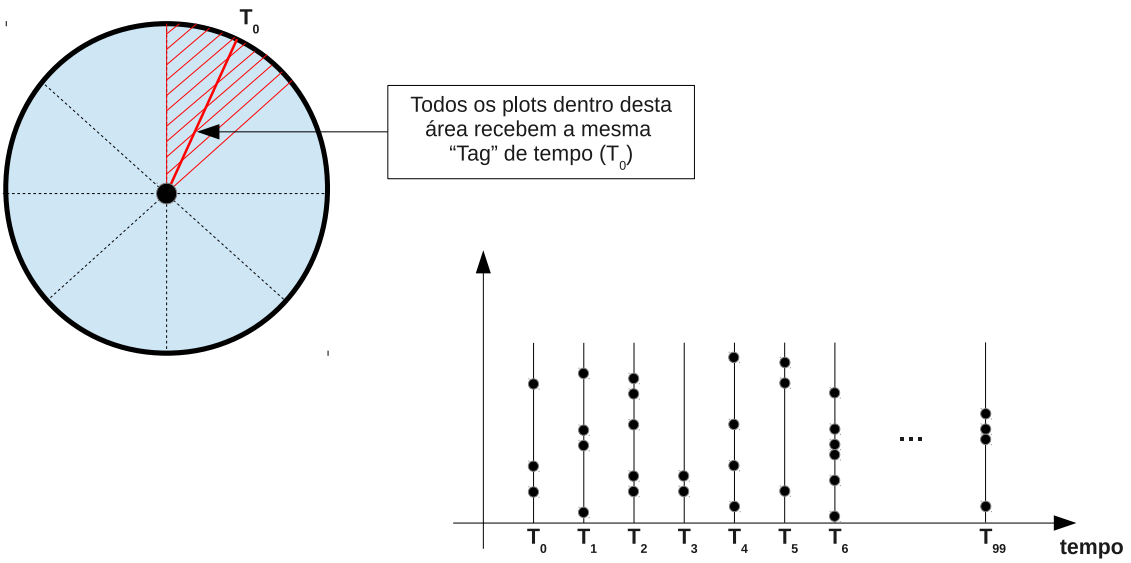

Figura 4.2: Processo de aproximação das marcações de tempos dos plots radar

Com relação à característica do movimento do alvo, um importante detalhe de implementação vale a pena ser mencionado: para o algoritmo proposto no método PPDE, se fazem necessários três plots iniciais por meio dos quais as informações a respeito da cinemática do alvo serão extraídas. Para o correto funcionamento do algoritmo, deve haver a garantia de que estes três plots pertençam ao alvo ao qual deseja-se realizar o acompanhamento. Para tanto, tais informações podem ser fornecidas através de outros sistemas de posicionamento como o GPS (*Global Positioning System*), por exemplo. Dentre as principais informações calculadas, destaca-se a velocidade máxima do alvo. A partir desta informação de velocidade, as arestas do grafo serão futuramente geradas.

Outro importante detalhe de implementação diz respeito ao conceito de janela de amostragem. Esta janela tem tamanho configurável no algoritmo implementado no método PPDE e diz respeito à quantidade de intervalos de tempo que serão levados em consideração para a formação dos grafos. Tomamos como exemplo uma janela de tamanho 20. Isto significa que na primeira execução do algoritmo, o mesmo irá aguardar 20 instantes de tempo (também configurável mas normalmente este instante de tempo corresponde a 0,01 seg) para que todos os dados da janela possam ser recebidos. Nas execuções posteriores, a cada instante de tempo, o método descarta o plot com a marcação de tempo mais antiga e processa o algoritmo incluindo os dados recebidos no instante de tempo mais recente, implementando um conceito de janela deslizante.

Uma vez que a janela de amostragem esteja completa, o primeiro estágio do algoritmo pode ser iniciado. Através da velocidade máxima calculada pelos três plots iniciais, serão executadas tentativas de criação de arestas entre todos os plots considerados. Cada plot carrega uma série de informações, dentre elas coordenadas (x, y) que identificam seu posicionamento num plano cartesiano. Dessa forma, dados dois plots A e B quaisquer, com a informação de velocidade máxima calculada anteriormente, é possível verificar se A consegue alcançar B. A tentativa de formação de arestas só é válida se, partir de um plot origem que possua marcação de tempo mais antiga que um plot destino, correspondendo a um caminho percorrido pelo alvo na direção passado-futuro. A Figura 4.3 ilustra o processo de formação de arestas entre os plots de uma janela qualquer a partir da velocidade máxima calculada por três pontos iniciais.

O método PPDE é útil principalmente em um cenário que utiliza vários sensores. Assim, a formação do grafo de plots descrita no parágrafo anterior pode ser realizada em paralelo, para cada sensor envolvido no processo. Os autores do PPDE, até a data de produção deste trabalho, não executaram o algoritmo em um ambiente paralelo, o que possivelmente traria enorme ganhos de desempenho. Ainda assim, o desempenho trazido pelo PPDE comparado ao PP trouxe ganhos

significativos, principalmente pela proposta do processamento em dois estágios, conforme podemos verificar em [4, 36].

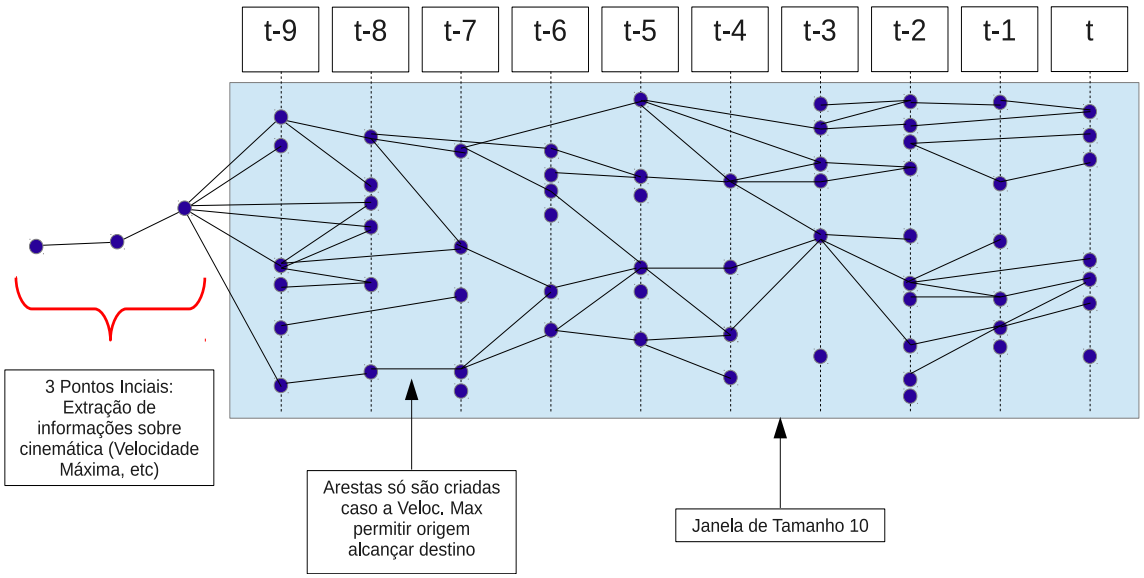

Figura 4.3: Processo de formação de arestas entre plots de uma janela a partir de plots iniciais

Tendo os Grafos de Plots (GP) sido estipulados para cada sensor, o PPDE constrói os Grafos de Pares de Plots (GPP). No GPP, cada aresta do GP se transforma em um vértice no GPP, conforme demostramos na Figura 3.5, que ilustra o primeiro estágio do PPDE. Com os grafos de pares de plots prontos, um algoritmo de otimização (Algoritmo de Dantzig) é aplicado visando buscar o caminho ótimo (caminho de menor custo). Neste ponto temos, para cada sensor, um conjunto de dados (vértices e arestas do grafo de par de plots) que fazem parte do caminho ótimo de cada grafo. De posse destes dados, consegue-se obter os vértices e as arestas do grafo de plot original, para cada sensor, finalizando o primeiro estágio.

O segundo estágio parte de um conjunto de plots que é resultado do primeiro estágio. Ou seja, a partir dos plots que fazem parte do melhor caminho calculado no primeiro estágio para cada sensor. Aqui, já não se faz a distinção entre plots de sensores distintos, utilizando-se os melhores plots de todos os sensores. Agora, processamento semelhante ao primeiro estágio será executado: um grafo de plots será criado, seguido do correspondente grafo de par de plots, onde posteriormente o Algoritmo de Dantzig será aplicado. Após esta otimização (aplicação de Dantzig), será encontrada a melhor hipótese de caminho para o alvo em questão. Tanto o primeiro quanto o segundo estágio são ilustrados nas Figuras 3.5 e 3.6, respectivamente.

4.3 Solução Desenvolvida

A presente seção é dedicada a explorar os detalhes inerentes à solução proposta. Como mencionado anteriormente, o objetivo principal deste trabalho é implementar um algoritmo de fusão de dados, aplicado ao acompanhamento de contatos, no modelo computacional Gamma.

Os métodos que configuram o estado da arte no que se refere à fusão de dados através da representação dos dados em grafos, possuem grande potencial para o paralelismo. Dentre estes métodos podemos destacar o PP [5] e o PPDE [4, 36], conforme abordamos no Capítulo 3. Especialmente o segundo método (PPDE), com a introdução de dois estágios de processamento, permite que a computação do caminho ótimo de todos os sensores envolvidos seja realizada de forma paralela. Esta característica dos métodos citados gerou a procura por modelos computacionais que permitissem a execução paralela de tarefas. Neste contexto, o formalismo Gamma apresenta-se como excelente possibilidade de obtenção do grau de paralelismo necessário, não somente a esta aplicação de fusão de dados, mas também para aplicações de diversas áreas. Além disso, implementações de problemas reais, como o trabalho em questão, tornam-se importantes mecanismos para a comunidade acadêmica para aprimoramento, identificação de vulnerabilidades, levantamento de pontos positivos e indicações de trabalhos futuros que tragam benefícios para implementações do modelo em questão.

4.3.1 O Multiconjunto

Iniciaremos a explanação do algoritmo proposto neste trabalho pela base de dados utilizada: o multiconjunto. Conforme abordado anteriormente, no modelo computacional Gamma, o multiconjunto é a única base de dados e foco de atuação de todas as reações existentes no programa. Todo e qualquer tipo de dado utilizado pelo programa em Gamma, deve estar representado no multiconjunto.

Os métodos de fusão de dados que constituem o estado da arte, e que foram utilizados como referência, utilizam-se de grafos para expressar os dados pertencentes à aplicação. Portanto, o problema inicial era encontrar uma forma de representação de grafos em Gamma. Diante disso, optou-se pela utilização de tuplas para representar os vértices e arestas dos grafos a serem utilizados, conforme ilustrado na Figura 4.4. Esta representação dos dados através de grafos também pode ser vista em [39] dentre outros autores. Vale ressaltar que utilizaremos um grafo dirigido de grau de saída N, onde N é o número de vértices. Ou seja, todas as arestas possuem um sentido e a quantidade de arestas que saem de um vértice qualquer é de no máximo N.

Assim, como cada plot utilizado em nossa aplicação será representado por um vértice, inicialmente as seguintes informações referentes aos plots foram necessárias:

- Identificador do sensor que realizou a aquisição do dado (*id_sensor*);
- Identificador do vértice. Trata-se de um número inteiro que identifica univocamente um vértice dentro do multiconjunto;
- Coordenadas (x, y) do plano cartesiano, referentes à posição da marcação do plot;
- Marcação de tempo atribuída pelo sensor; e
- Um campo *"marca"* utilizado pelas reações como artifício para identificar se determinado elemento foi utilizado ou não em determinado caso. Veremos maiores detalhes da utilização deste campo na Subseção 4.3.2.

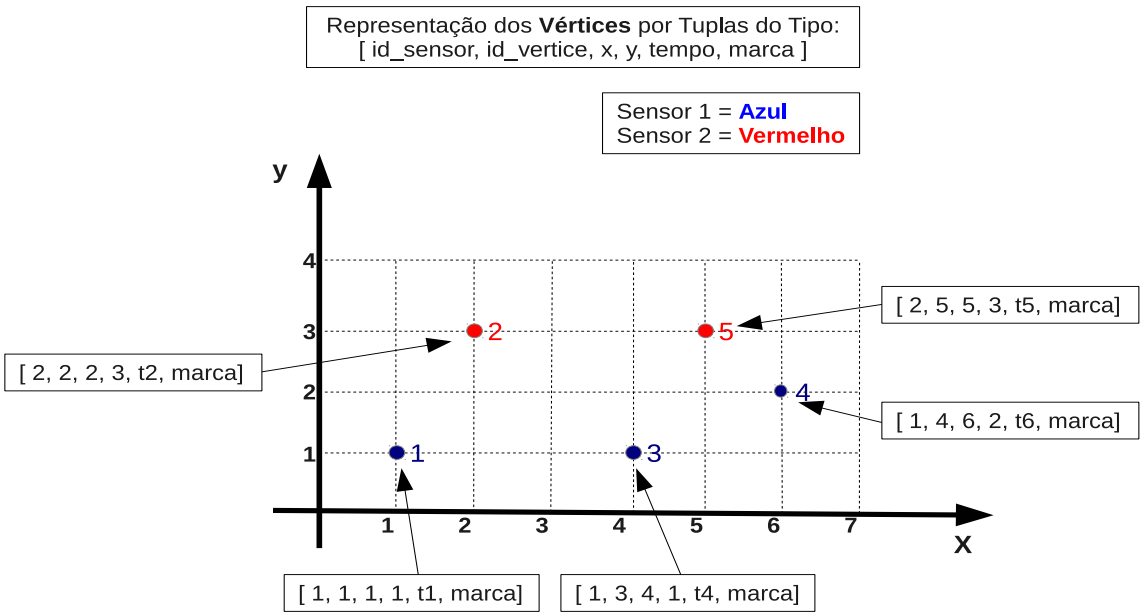

Figura 4.4: Representação de vértices de grafos em tuplas

Além dos vértices citados acima, o multiconjunto inicial possui outro elemento básico: tuplas do tipo arestas. Estas, possuem as seguintes informações, inicialmente:

- Identificador do sensor que realizou a aquisição do dado (*id_sensor*);

- Identificador do vértice de origem (*id_vertice_origem*); e
- Identificador do vértice de destino (*id_vertice_destino*).

Tanto as tuplas que representam arestas quantos as que representam vértices podem ter sua quantidade de elementos alterada em tempo de execução. Isto é possível pela aplicação do esquema de programação *TROPES* que foi abordado no Capítulo 2, mais especificamente os padrões *Reducer* e *Expander* que reduzem e aumentam a quantidade de elementos do multiconjunto, respectivamente. Estes conceitos são aplicáveis ao multiconjunto, porém neste trabalho foram utilizados no escopo de uma tupla, alterando o tamanho inicial das mesmas. Vale mencionar que, por características da implementação de Gamma utilizada, os únicos valores suportados pelo multiconjunto são números inteiros ou tuplas formadas por números inteiros. Como veremos nas seções seguintes, algumas vezes um número inteiro que não pertence à uma tupla é inserido no multiconjunto para permitir a execução de algumas reações.

Diante dos argumentos expostos, verificamos que, inicialmente, o multiconjunto é constituído de um grande conjunto de tuplas que representam vértices e arestas de alguns grafos (um grafo por sensor). É importante mencionarmos que os grafos representados inicialmente são grafos completos, ou seja, existem arestas de todos os vértices (plots) para todos os outros vértices adquiridos por um mesmo sensor. Essa característica da solução deve-se ao fato da impossibilidade de estipular uma condição de parada para a computação de uma reação que deveria criar as arestas do grafo. Em outras palavras, seria desejável implementar uma reação que criasse as arestas necessárias caso dois vértices cumprissem as condições necessárias à criação de uma aresta. Conforme veremos na seção a seguir, 4.3.2, não foi possível estabelecer os critérios necessários à criação desta reação, o que levou a solução proposta a partir de um grafo completo.

4.3.2 Algoritmo proposto

Nesta subseção abordaremos as características inerentes ao algoritmo proposto em nossa solução. Inicialmente apresentaremos as condições de parada necessárias para a implementação de uma reação, o que irá justificar a utilização de um grafo completo como ponto de partida de nossa solução. A utilização de um grafo completo também foi realizada em [40], dentre outros autores. Posteriormente, verificaremos maiores detalhes a respeito do método utilizado para o cálculo do melhor caminho. Finalmente, visualizaremos o algoritmo como um todo, do ponto de vista do conjunto de reações que compõem a solução desenvolvida.

4.3.2.1 Condições de Parada

Numa reação, um importante aspecto a ser considerado é a **condição de parada** da computação. Ou seja, deve existir algum estado terminal da computação para cada reação envolvida. Do caso contrário, a reação pode vir a ter sua condição de reação sempre satisfeita, o que acarretaria na execução infinita desta reação. Imaginemos, por exemplo, o seguinte código hipotético em Gamma:

$$\begin{aligned} & R\ \{1,2,3,4\} \\ & \quad where \\ & R = replace\ \ x \\ & \quad by\ \ x+1 \\ & \quad if\ \ x \geq y \end{aligned} \tag{4.4}$$

A condição de reação $x \geq y$, no exemplo acima, sempre estará satisfeita, já que mesmo com a ação de "somar um" executada, sempre existirão dois elementos pelo menos iguais no multiconjunto. Ou seja, nenhuma **condição de parada** da execução desta reação será possível de ser atingida.

Um artifício bastante utilizado em programação Gamma e, muito utilizado em nossa solução, é implementar a ação a ser executada de forma a modificar o multiconjunto, com o objetivo de impossibilitar que a condição de reação seja verdadeira indefinidamente, criando assim uma condição de parada. Por exemplo, tomemos o seguinte código Gamma, baseado no exemplo anterior 4.4:

$$\begin{aligned} & R\ \{1,2,3,4\} \\ & \quad where \\ & R = replace\ \ x,\ \ y \\ & \quad by\ \ x+1 \\ & \quad if\ \ x \geq y \end{aligned} \tag{4.5}$$

Aqui, utilizou-se o padrão *Reducer* do TROPES, como vimos na Subseção 2.2.2 pois, sempre que existirem dois elementos x e y no multiconjunto que satisfizerem a condição $x \geq y$, o menor elemento da comparação será excluído (y) enquanto que o maior será mantido acrescido de uma unidade. É importante notar que a cada execução da reação, um elemento será excluído do multiconjunto, de forma que chegará o instante em que não será possível a execução da reação pois o multiconjunto será composto por um único elemento, o que configura a reação ter encontrado sua **condição de parada**.

Voltando para nossa implementação, ao final da Seção 4.3.1 mencionamos a necessidade de utilização de um grafo completo no multiconjunto inicial pela impossibilidade de alcançar uma condição de parada para a reação que viria a criar as arestas do grafo. Como mencionamos anteriormente, o grau de saída do grafo é N, onde N é o número de vértices existentes inicialmente. Assim, é impossível prever quantas arestas serão criadas a partir de um vértice qualquer. Em outras palavras, inicialmente possuímos N plots, que serão candidatos a vértices no grafo inicial. A criação de uma aresta de um vértice A, para um vértice B, dada uma velocidade máxima, ocorre quando:

$$V_{max} \geq \frac{d_{A,B}}{t_B - t_A}$$

Onde:

- V_{max} é a velocidade máxima calculada através dos três plots iniciais;
- $d_{A,B}$ é a distância entre os vértices A e B, dada por: $d_{A,B} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$; e
- t_B e t_A são as marcas de tempo atribuídas pelo sensor aos vértices (plots) A e B, respectivamente.

Dessa maneira, caso o critério acima seja satisfeito para dois plots A e B, então uma aresta de A a B deverá ser criada. Porém, um vértice A qualquer poderá possuir arestas para todos os outros vértices com marcações de tempo superiores a este. Deste modo, deveria existir uma reação responsável por criar arestas entre dois vértices quaisquer. Porém, em Gamma, as reações ocorrem livremente sob o multiconjunto e a condição expressa na reação deve ser também responsável por parar a computação desta reação, impedindo que esta tenha sua condição de execução sempre satisfeita. Assim, o desafio aqui é estipular uma **condição de parada** para uma reação que visa criar arestas entre dois nós. Para criar tal reação, foram realizados testes para alterar um parâmetro (componente "marca") das tuplas do tipo vértice, foram realizados outros estudos visando a modificação do multiconjunto porém não obtivemos nenhum resultado positivo, basicamente pela característica do grau de saída do grafo ser diferente de um. Em outras palavras, não foi possível obter uma condição de execução para impedir que uma mesma aresta fosse inserida várias vezes no multiconjunto. Por estes motivos, foi necessária a inicialização do algoritmo da solução proposta a partir de um grafo completo.

Assim, o enfoque das três primeiras reações de nossa solução, não é criar arestas necessárias, mas sim retirar as arestas desnecessárias de um grafo completo.

4.3.2.2 Cálculo do Melhor Caminho

Antes de passarmos para maiores detalhes sobre o conjunto de reações envolvidas na aplicação, um aspecto importante a ser abordado é o cálculo do melhor caminho no grafo.

Uma vez retiradas as arestas desnecessárias, a reação R4 responsabiliza-se por atribuir um custo às todas as arestas restantes do grafo, conforme veremos na Subseção 4.3.2.3. Assim, ao final da execução desta reação, possuímos um multiconjunto tal que todas as tuplas do tipo aresta possuem um parâmetro de custo (baseado em informações de velocidade) associado. Em linhas gerais, desejamos encontrar o caminho de menor custo, que corresponderá aquele cuja a variação da velocidade por trecho (aresta) foi a mais próxima possível da velocidade estimada dada a informação de aceleração do grafo. Este parâmetro de aceleração é calculado através dos três plots iniciais fornecidos, assim como a velocidade máxima mencionada anteriormente. Ou seja, para cada aresta foi atribuído um custo relativo ao quão próximo esta está do valor de velocidade tido como o ideal calculado.

O método implementado pelas reações envolvidas no cálculo do melhor caminho no grafo foi baseado em [39] e consiste, em linhas gerais, na seguintes etapas:

- inicialmente, para cada aresta adjacente ao vértice origem, é calculado o parâmetro "somatório dos custos"(S), como sendo igual ao próprio custo da aresta;

- Após isso, para as demais arestas envolvidas, calcula-se S como sendo o somatório entre o custo da própria aresta acrescido do custo da aresta imediatamente anterior em um caminho qualquer. Para isso a reação R6 busca no multiconjunto arestas que são vizinhas em um caminho, ou seja, dados três vértices A, B e C, as arestas (A,B) e (B,C) são vizinhas no caminho {A,B,C} e (A,B) é a aresta imediatamente anterior a aresta (B,C) no caminho {A, B, C};

- De todas as arestas que chegam a um determinado vértice, somente a de menor custo é mantida; e

- Por fim, são excluídas as arestas que não fazem parte do melhor caminho.

Este método calcula o custo total de um caminho através da acumulação de custos, de forma que ao analisarmos o vértice destino, após a aplicação do método, a aresta que chega a este vértice possui o campo S como sendo o valor acumulado com os custos de todas as arestas do melhor caminho.

Na Figura 4.5, informações sobre os dois primeiros passos do método são fornecidas. O vértice tido como origem é o terceiro plot fornecido inicialmente, a partir

do qual existe a confiança de que seja uma marcação real do alvo cujo acompanhamento necessita-se gerar. Dessa forma, a partir do vértice origem, calcula-se o somatório do custo (S), como sendo o próprio custo da aresta em questão, para todas as arestas adjacentes ao vértice origem, constituindo o primeiro passo do método para obtenção do melhor caminho. Nesta ilustração, os vértices adjacentes ao vértice origem encontram-se marcados de vermelho.

Uma vez calculados os valores de S para as arestas adjacentes ao vértice origem, a reação R6 é responsável por atualizar o campo S de todas as outras arestas do grafo, identificadas em azul na Figura 4.5, o que configura o segundo passo do método. Dessa forma, dada uma aresta qualquer não adjacente ao vértice origem, o cálculo de seu somatório de custos será definido pela soma do custo da aresta em questão e do campo S da aresta imediatamente anterior a esta em um dado caminho, desde que esta já tenha tido seu campo S calculado. Na Figura 4.5, as arestas que interligam nós com marca de tempo $t-4$ e $t-3$ possuem arestas imediatamente anteriores (arestas que partem do vértice origem) com o campo S calculado no primeiro passo do método. Dessa forma para estas arestas que partem de $t-4$ em direção a $t-3$, o cálculo mencionado, $S_{atual} = Custo_{atual} + S_{anterior}$, é válido. Como as reações em Gamma vão sendo executadas na medida em que os dados (multiconjunto) estão prontos para serem executados (condição de reação satisfeita), o cálculo de S das demais arestas vai ocorrendo de naturalmente como resultado, ainda, da execução de R6.

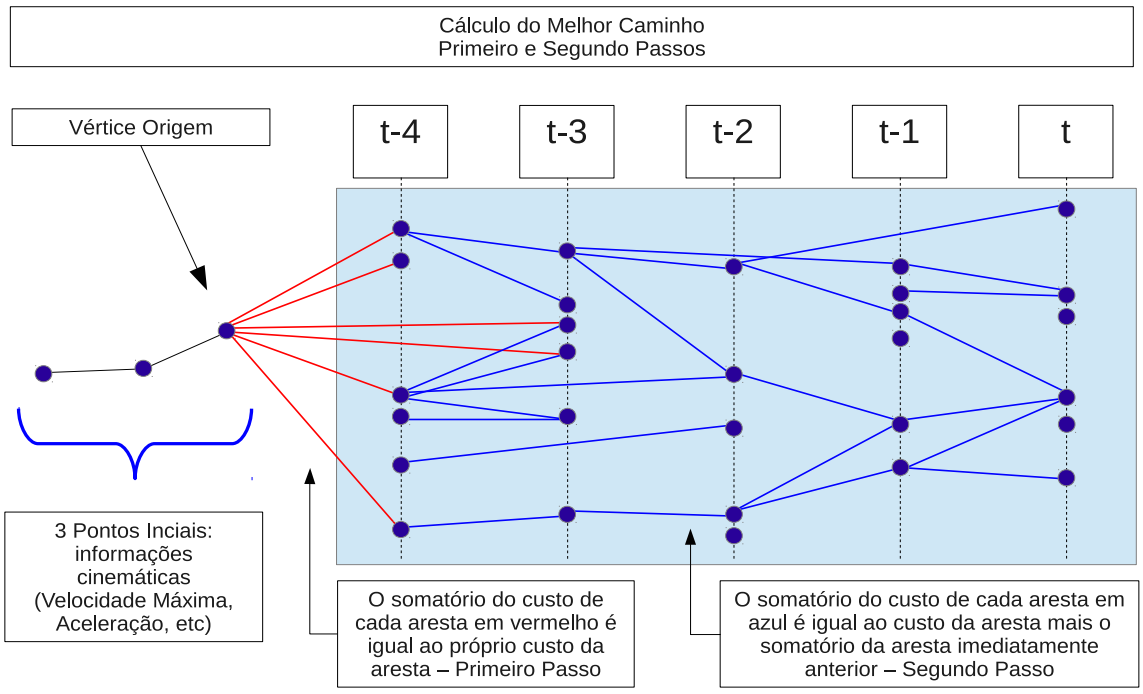

Figura 4.5: Primeiro e segundo passos do algoritmo de cálculo do melhor caminho no grafo

Já na Figura 4.6, ficam evidenciadas as ações executadas nos passos 3 e 4 através de um exemplo hipotético onde as arestas em negro fariam parte do caminho de menor custo. Ao finalizarmos o passo 2, a próxima tarefa é excluir as arestas desnecessárias e encontrar o melhor caminho. O passo 3 consiste em excluir do multiconjunto parte das arestas desnecessárias. A ação executada neste passo 3 seria genericamente a seguinte: para todo vértice com grau de chegada maior que 1, manter somente a aresta que chega com menor custo acumulado (S). Assim, na Figura 4.6, as arestas em vermelho seriam excluídas pois não são as de menor valor de S que chegam aos seus respectivos vértices destino.

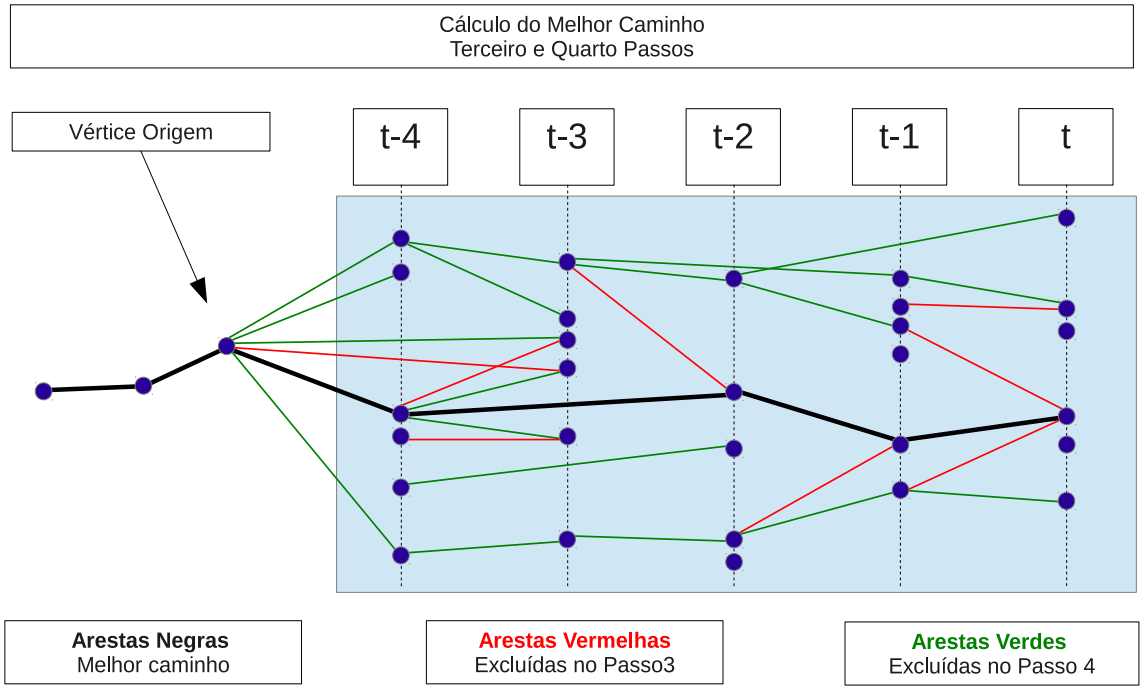

Figura 4.6: Terceiro e quarto passos do algoritmo de cálculo do melhor caminho no grafo

É importante salientar que neste momento, já temos condições de identificar o caminho de menor custo, através da realização do percurso inverso no grafo, no sentido destino-origem, uma vez que para cada nó do grafo só chega uma aresta. Assim, durante este percurso, tal caminho vai sendo marcado (tuplas do tipo aresta vão sendo "marcadas"). No exemplo da Figura 4.6, este caminho hipotético seria identificado pelas arestas negras.

Por fim, as arestas que não foram marcadas por pertencerem ao melhor caminho (arestas em verde), podem ser excluídas, finalizando o método em seu passo quatro.

4.3.2.3 Detalhamento das Reações

O objetivo desta subseção é apresentar uma visão global, do ponto de vista do conjunto de reações desenvolvido, sobre o algoritmo da solução aqui proposta. De uma maneira mais ampla, o algoritmo possui como entrada um grafo completo para cada conjunto de plots separados por sensor. Ou seja, se no cenário utilizado possuirmos três sensores radares, teremos três grafos completos, onde os vértices são os plots do radar. O algoritmo processa estes dados em dois estágios, conforme veremos na Seção 4.3.3, sendo que o primeiro estágio fica responsável por fornecer como resposta um conjunto de vértices selecionados, por sensor, e que fazem parte do melhor caminho calculado para cada sensor. Já no segundo estágio, a entrada é constituída também por um grafo completo. Porém, os vértices deste grafo são os plots que formam o conjunto de vértices selecionados de cada sensor no primeiro estágio. Ou seja, no segundo estágio o grafo inicial é formado por vértices de diversos sensores.

Antes de apresentarmos cada reação existente, é necessário mencionar que, pela característica do problema a ser abordado, as transformações no multiconjunto em questão devem ser realizadas uma a uma de maneira sequencial. Dessa maneira, um conjunto de 16 reações são executadas sequencialmente através da utilização do operador ";" conforme abordado no Capítulo 2. Entretanto, caso exista mais de um sensor envolvido, este conjunto de reações poderá executar em paralelo com outros conjuntos de reações semelhantes, conforme veremos mais a frente.

Na Figura 4.7, podemos verificar de uma maneira resumida todas as 16 reações existentes e suas funcionalidades principais. Logo em seguida verificaremos maiores informações sobre as tarefas específicas de cada reação. Conforme ilustrado nesta figura, o conjunto de 16 reações existentes pode ser agrupado em 5 blocos funcionais. O conjunto inicial formado pelas três primeiras reações é responsável por, dado um grafo completo, excluir algumas arestas desnecessárias, como por exemplo aquelas que saem e chegam ao mesmo vértice, se existirem. Outro tipo de arestas indesejáveis são aquelas que fazem referência ao passado, uma vez que partem de um vértice com marca de tempo x e chegam a um vértice com marca de tempo y, tal que $y < x$.

O segundo bloco funcional é composto por uma única reação com a finalidade de atribuir o parâmetro de custo inerente a cada aresta baseado em informações cinemáticas do alvo (velocidade). Já o terceiro bloco, abordado em maiores detalhes na Subseção 4.3.2.2, implementa o método para cálculo do melhor caminho no grafo especificado. Após o cálculo do melhor caminho, os vértices que não fazem parte deste podem ser descartados, conforme implementado no quarto bloco funcional. Por fim, é necessária uma série de transformações no multiconjunto visando a preparação para o próximo estágio. Este conjunto de ações constituem o quinto e último bloco.

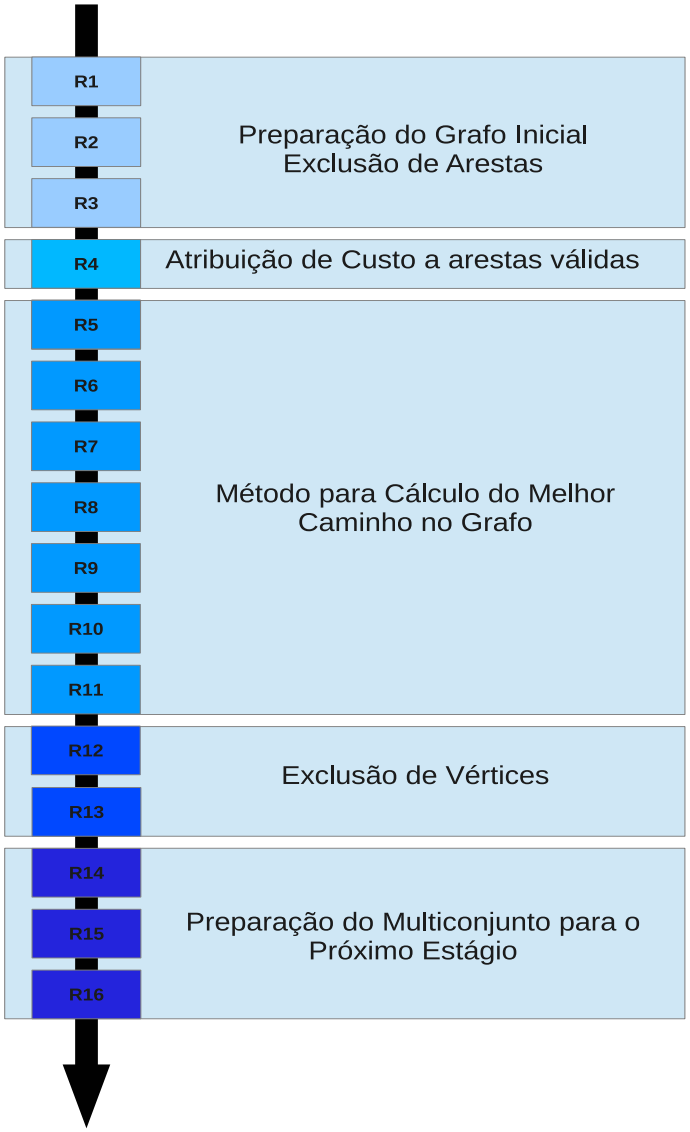

Figura 4.7: Generalização do algoritmo da solução proposta

Após termos abordado os cinco blocos que agrupam reações com tarefas afins, finalizaremos esta seção com uma descrição de cada reação envolvida, conforme podemos observar abaixo:

1. A primeira reação existente, R1, é responsável pela exclusão das arestas que saem e chegam a um mesmo vértice. Para fins de ilustração, iremos reproduzir abaixo esta reação e explica-la em melhores detalhes, as demais reações que compõem o algoritmo estarão no apêndice A, assim como o multiconjunto inicial.

$R1 = replace\ \ [ids, ida, idb], [ids1, ida1, idb1]$
$by\ \ [ids1, ida1, idb1]$
$if\ \ ids == ids1\ \ and\ \ ida == idb\ \ and\ \ ida! = 1$

Aqui, R1 compara inicialmente duas tuplas do tipo aresta formada por 3 elementos, $[ids, ida, idb]$, onde ids é o identificador do sensor a que esta aresta pertence, ida corresponde ao vértice origem desta aresta e idb o vértice destino. A ação desta reação é excluir uma das arestas, uma vez que no parâmetro *replace* temos duas tuplas ao passo que no parâmetro *by* possuímos somente uma. Conforme podemos ver na terceira linha da reação, o parâmetro *if* explicita a condição que tem que ser cumprida para que haja a exclusão de uma aresta. Tal condição é expressa por $ids == ids1 \ and \ ida == idb \ and \ ida! = 1$. Ou seja, caso ida for igual a idb, a aresta sai e chega a um mesmo vértice e por isso pode ser excluída. O último parâmetro, $ida! = 1$, é inserido para evitar a exclusão de uma aresta auxiliar utilizada em comparações com outras reações;

2. A segunda reação exclui todas as arestas que fazem referência ao passado pois interligam vértices cujo destino é mais antigo que a origem. Isto está diretamente relacionado a aplicação de fusão de dados, uma vez que um alvo não pode percorrer um caminho que remete ao passado;

3. R3 testa todas as arestas com relação a consistência cinemática do movimento do alvo, verificando se é possível ou não a existência de tal aresta. As arestas que não satisfizerem tal condição são excluídas do multiconjunto;

4. Uma vez excluídas as arestas inválidas, R4 transforma as arestas válidas incluíndo o custo unitário para cada uma delas. Tal custo baseia-se em informações de velocidade. É importante não confundir este custo unitário da aresta, com o campo S, que trata do somatório dos custos de parte de um caminho até determinado vértice;

5. R5 é o primeiro passo do método que calcula o melhor caminho. Nesta reação o campo S (somatório dos custos) de todas as arestas adjacentes ao vértice origem é calculado, como vimos anteriormente, na Subseção 4.3.2.2;

6. A sexta reação calcula e atribui o somatório dos custos (campo S), para as demais arestas envolvidas;

7. R7 tem por objetivo excluir todas as arestas que chegam ao mesmo vértice, exceto aquela de menor valor de S;

8. R8 insere no multiconjunto outro tipo de tupla: identificadores $[ids, idv]$, onde ids é o identificador do sensor e idv o identificador do vértice. Nesta reação são inseridos no multiconjunto tuplas do tipo "identificador" somente para aqueles vértices que são origem ou destino de alguma aresta. Esta operação

será importante para as reações seguintes, pois será necessário identificar o vértice de maior id;

9. Em R9, dentre as tuplas do tipo "identificador" incluídas na reação anterior, é selecionada a de maior id;

10. Tendo a informação do vértice alcançável de maior id, R10 caminha, a partir deste vértice, no sentido destino-origem marcando arestas do melhor caminho, uma vez que só chega uma aresta a cada vértice;

11. Tendo marcado as arestas do melhor caminho na reação anterior, as demais arestas, não marcadas, são excluídas do multiconjunto como resultado da execução de R11;

12. R12 marca todos os vértices que fazem parte do melhor caminho. Ou seja, como neste ponto da computação todas as arestas do multiconjunto fazem parte do melhor caminho, todos os vértices que são origem ou destino de alguma aresta são marcados em R12;

13. Em R13, todos os nós marcados em R12 são excluídos, ou seja, exclui-se todos os nós que não fazem parte do melhor caminho;

14. Como agora o multiconjunto é constituído de algumas arestas (arestas do melhor caminho) e um conjunto de vértices de melhor caminho, posso excluir todas estas arestas pois, na realidade, a informação necessária tanto para outro estágio ou para o fim do algoritmo são os vértices. Assim R14 exclui todas as arestas visando o fim do algoritmo ou o segundo estágio, como veremos mais adiante;

15. R15 nada mais faz do que "desmarcar" os vértices selecionados visando outro estágio ou o fim do algoritmo; e

16. Finalmente, o elemento auxiliar (tupla auxiliar utilizada na comparação de algumas reações) é excluído.

Por fim, é necessário mencionar que algumas funções matemáticas não suportadas na implementação de Gamma utilizada tiveram que ser inseridas no código "C" produzido pelo compilador Gamma manualmente, como por exemplo a função de radiciação. Ou seja, o código fonte do compilador teve de ser alterado. Dessa forma, nas reações R3 e R4, no cálculo de custo e na verificação da validade de determinada aresta foi necessário alterar o código gerado pelo compilador Gamma após a análise sintática, mais especificamente os arquivos *run.c* e *run.gpu* conforme foi ilustrado na Figura 4.1.

4.3.3 Implementação dos Estágios

Conforme verificamos no Capítulo 3, mais especificamente na Seção 3.3.3, a solucão proposta por BARBOSA [4] baseia-se em dois estágios de processamento. A ideia de dividir o processamento em dois estágios está intimamente ligada à possibilidade de processar os grafos para cada sensor em um primeiro estágio e, após a escolha dos melhores plots para cada sensor, executar o segundo estágio que busca os plots ideais em um grafo cujos vértices correspondem aos plots selecionados de todos os sensores. Portanto, a computação envolvida no primeiro estágio do PPDE tem grande potencial para ser executado em um ambiente paralelo.

Na solução proposta, são implementados dois estágios, baseados no conceito introduzido pelo PPDE. A computação executada em cada estágio é semelhante e composta por 16 reações em cada estágio, conforme vimos anteriormente. O conjunto de reações para o primeiro e segundo estágio podem ser verificados no Apêndice A.

Na Figura 4.8, são ilustrados os dois estágios do algoritmo proposto neste trabalho para dois sensores distintos. Os plots do sensor 1 são representados por losangos negros enquanto que os plots do segundo sensor aparecem como quadrados vermelhos na ilustração. É importante perceber que o cálculo do melhor caminho para o primeiro estágio é realizado por sensor. Após a linha vermelha pontilhada o segundo estágio é iniciado com os plots selecionados dos dois sensores. Neste ponto, o grafo é gerado com plots pertencentes a todos os sensores envolvidos que, no caso do exemplo em questão, constituem um total de dois sensores. O final do segundo estágio é representado por um caminho composto pelos plots que correspondem à melhor hipótese de caminho assumido pelo alvo ao qual necessita-se acompanhar e monitorar.

Vale ressaltar que os dois estágios descritos foram implementados em códigos Gamma distintos, pela impossibilidade de especificação de arestas mencionadas na Seção 4.3.2.1. Ou seja, ao final do primeiro estágio, todos os plots selecionados para cada sensor devem formar novamente um novo grafo. Para tanto, deveria existir uma reação, ou conjunto de reações, que criasse tuplas do tipo aresta, o que não foi implementado pela impossibilidade de estipulação de condições de parada para as reações necessárias. Portanto, para permitir a execução do algoritmo em seus dois estágios existentes, foi implementado um código em C, que, dentre outras coisas, processa a saída do primeiro estágio e fornece tais dados para o segundo estágio do algoritmo, conforme veremos a seguir.

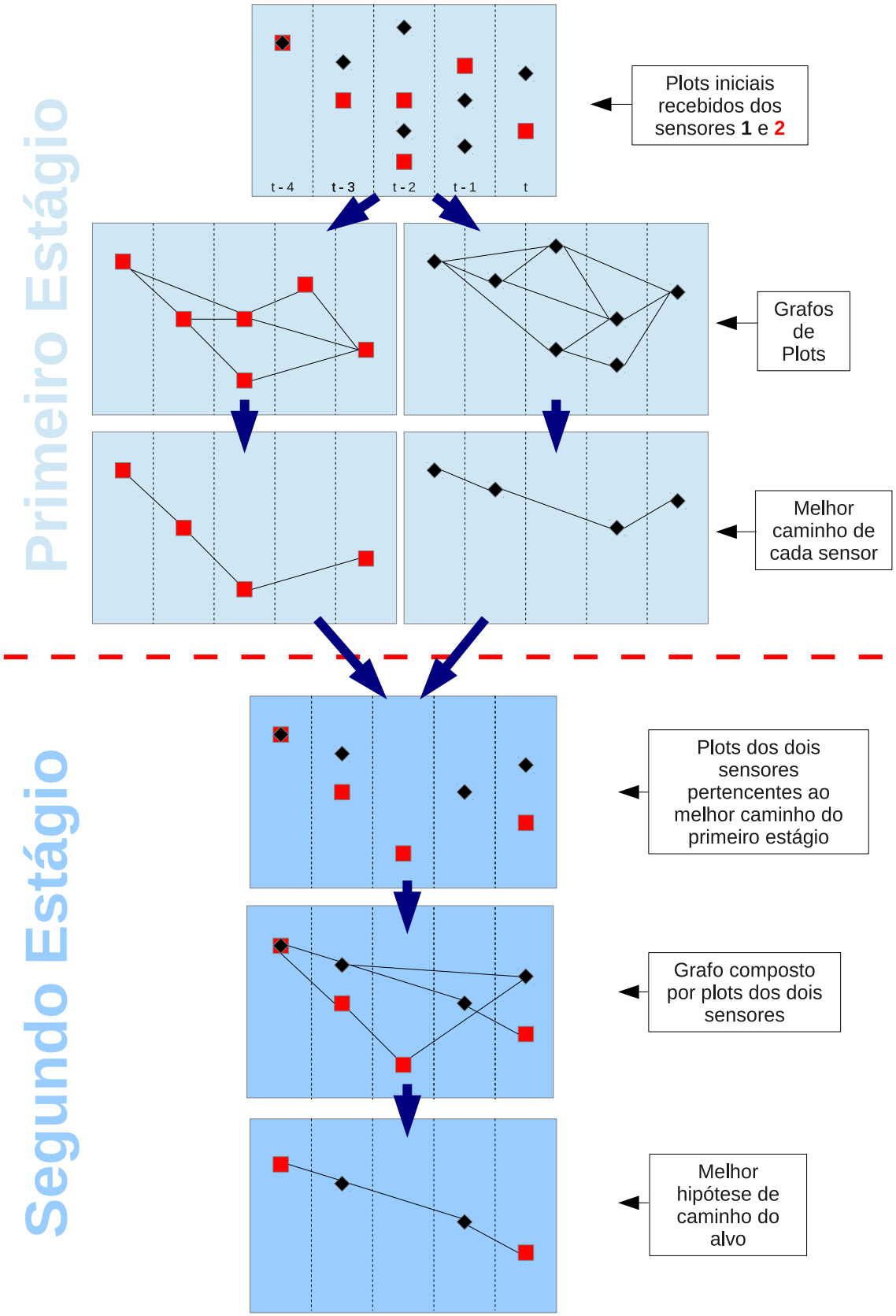

Figura 4.8: Visualização dos dois estágios da solução

4.3.4 Dinâmica Iterativa

Conforme abordamos na subseção anterior, os códigos referentes ao primeiro e ao segundo estágio de nosso algoritmo encontram-se implementados em códigos fonte-Gamma independentes. Dessa forma, para fornecer uma aplicação iterativa, ou seja, uma aplicação que execute os dois estágios e produza um resultado sem a chamada explícita do programador, foi necessária a adoção de um código fonte adicional. Tal código fonte foi escrito em linguagem C e possui algumas características e funcionalidades específicas como veremos a seguir:

- **Link entre os estágios**: Dentre outros pontos, o código atua entre o final do primeiro estágio e início do segundo. O resultado do primeiro estágio consiste num conjunto de tuplas do tipo vértice, ou seja, conjunto de plots. Para início do segundo estágio, da mesma forma que o primeiro, necessita-se de um grafo completo. Assim, a partir da leitura de um arquivo texto contendo os plots do primeiro estágio, o código gera as tuplas do tipo aresta necessárias para o segundo estágio e executa uma chamada ao código do segundo estágio; e

- **Produção de novos dados e início de outra execução**: Ao final do segundo estágio, os valores são escritos em um arquivo texto, formatados para uma nova execução e novos plots são fornecidos. Em outras palavras, ao final do segundo estágio teremos um conjunto de plots que corresponde à hipótese de melhor caminho para o alvo. Para simular a recepção de novos dados radar, o que configuraria uma nova passagem pelo algoritmo, alguns novos dados são fornecidos e são combinados com os dados resultantes do segundo estágio para darem início a uma nova execução (início do primeiro estágio novamente). Para início de uma novo passo do algoritmo (chamada ao primeiro estágio), o código mencionado realiza uma nova chamada ao código referente ao primeiro estágio, após os dados (conjunto de vértices e arestas) estarem prontos para uma nova execução.

Assim, para permitir a execução das funcionalidades acima mencionadas, algumas alterações nos códigos gerados pelo compilador Gamma foram necessárias:

- Como resultado do primeiro estágio, a função que escrevia o resultado da computação para o terminal foi redirecionada para um arquivo texto; e

- O compilador Gamma lia os dados iniciais do próprio arquivo fonte, fazendo com que estas informações fossem carregadas em definitivo no código C gerado pelo compilador Gamma. Dessa maneira, outra alteração necessária foi fazer com que o código C resultante do compilador Gamma passasse a popular as suas estruturas de dados internas ("Bags", [7]) com informações lidas a partir de um arquivo texto.

4.3.5 Abordagens Sequencial, Paralela e Híbrida

Para melhor visualizar como a implementação de Gamma utilizada trata o paralelismo ao nível de execução de suas reações, implementamos o algoritmo de fusão de dados em Gamma em três abordagens distintas que diferem na forma como o programador pode expressar seu paralelismo. Maiores detalhes poderão ser verificados no capítulo seguinte, mais especificamente na Seção 5.2, onde também são apresentados os resultados obtidos para estas três abordagens.

Em todas as três abordagens, o conjunto de 16 reações que corresponde ao código do algoritmo proposto continua mantido de forma sequencial para execução, ou seja, $R1; R2; R3; ...; R16$, onde o ";" corresponde ao operador de execução sequencial. De acordo com a exposição de fatos anteriores, esta sequencialidade da solução está intimamente relacionada às necessidades impostas pelo algoritmo de fusão de dados, onde cada transformação no multiconjunto deve ser realizada após a outra. Porém, com a implementação dos dois estágios propostos em [4, 36], surge a possibilidade de execução paralela das operações para cada sensor no primeiro estágio do algoritmo. Dessa forma, os casos de teste propostos na Seção 5.2 computam o tempo de execução somente do primeiro estágio do algoritmo, uma vez que, além do segundo estágio das três abordagens ser idêntico, os casos de teste mencionados foram submetidos remotamente à execução em um ambiente que possui hardware paralelo, assim, a utilização do código C necessário à execução abordado na Subseção 4.3.4, seria impossível de ser executado.

Na primeira delas, chamada de ***Abordagem Sequencial*** o conjunto de reações necessárias para cada sensor é executado de maneira sequencial. Ou seja, verificando o exemplo da Figura 4.9 onde o algoritmo processa dados provenientes de três sensores, o bloco de reações do terceiro sensor só é executado após o término do segundo bloco, que trata dados do sensor 2, e este só será colocado para execução após a finalização da computação necessária ao bloco de reações que processa dados do primeiro sensor, o bloco 1.

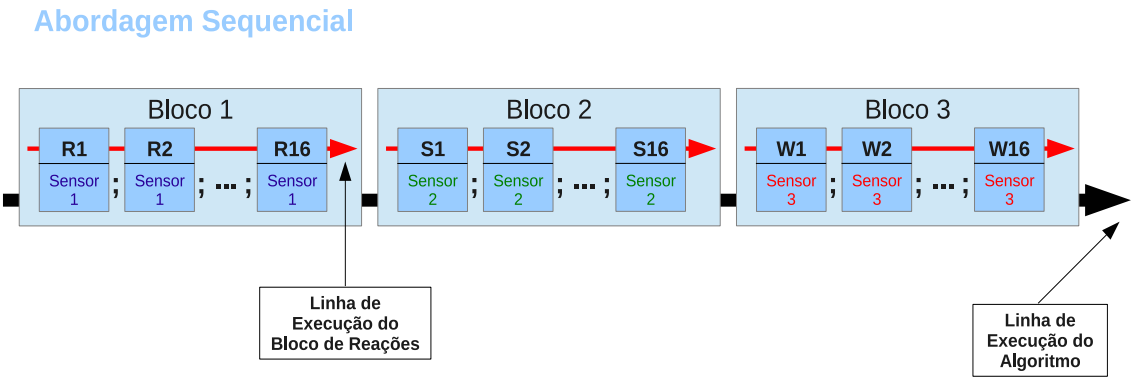

Figura 4.9: Abordagem sequencial

Já na ***Abordagem Paralela***, cada bloco de reações necessárias a cada sensor é posto a executar de forma paralela. Ou seja, como podemos verificar na Figura 4.10, considerando como "*Bloco 1*" o conjunto de reações referentes ao primeiro sensor, "*Bloco 2*" o bloco do segundo sensor e "*Bloco 3*" o bloco de reações do terceiro sensor, esta abordagem procura fazer $Bloco1 \mid Bloco2 \mid Bloco3$, onde o símbolo "|" corresponde ao operador de execução paralela, conforme visto na Subseção 2.3.1.2.

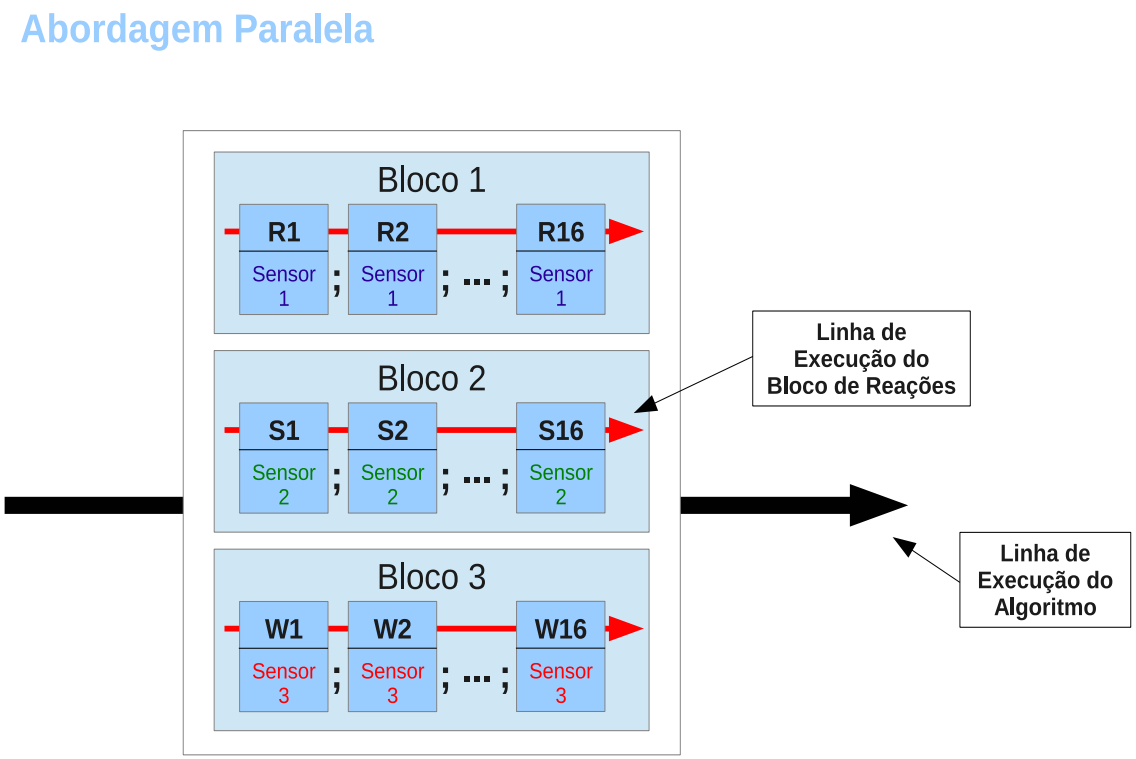

Figura 4.10: Abordagem paralela

Por fim, a ***Abordagem Híbrida*** expressa o paralelismo de forma mais elegante e eficiente conforme iremos demonstrar com os resultados que serão apresentados na Seção 5.3 e no exemplo da Figura 4.11. Aqui, existe um único "Bloco" de reações que são executadas sequencialmente mediante a utilização do operador de composição ";". Este bloco de reações é idêntico aos blocos existentes para cada sensor, que foram citados nas duas abordagens anteriores. Porém, para cada reação, uma das condições de execução da mesma é somente permitir que reajam dados (vértices ou arestas) advindos de um mesmo sensor. Assim, esta abordagem só permite a ocorrência de reações com elementos que possuam o campo Id_{sensor} idênticos. Com isso, uma mesma reação pode tratar os dados adquiridos por todos os sensores, com a ressalva de que a cada instância de execução desta reação, um conjunto de

elementos de um mesmo sensor qualquer estará sendo manipulado.

Por exemplo, dado um conjunto de reações $\{R1, R2, ..., R16\}$ e considerando a abordagem *Híbrida* utilizando dados advindos de três sensores, inicialmente teríamos uma instância de $R1$ reagindo os elementos advindos do sensor 1, enquanto outra instância de $R1$ poderia tratar dados pertencentes ao sensor 2 e assim sucessivamente. Ao final da execução de todas as instâncias de $R1$, a reação $R2$ iniciaria sua execução, seguida de $R3$, $R4$ até finalizar o algoritmo pela execução de $R16$. Dessa forma, podemos considerar que o paralelismo da abordagem *Híbrida* se dá de maneira intra reação.

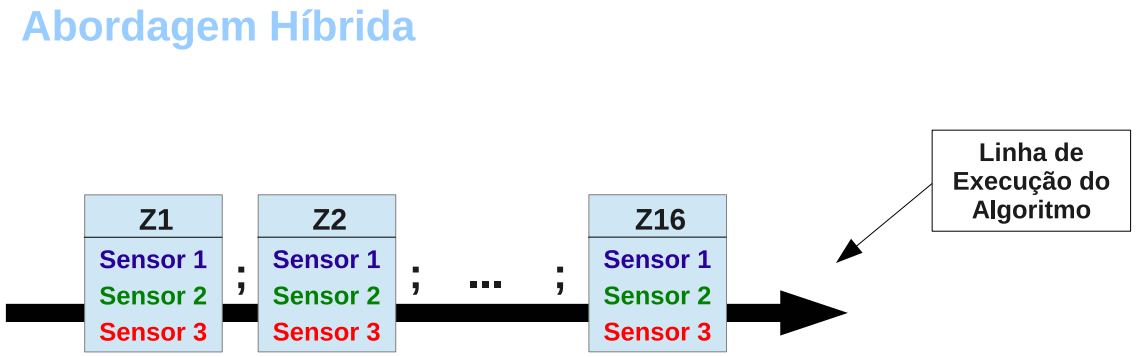

Figura 4.11: Abordagem híbrida

4.3.6 Dificuldades Encontradas

Após apresentarmos as características da solução proposta neste trabalho, nesta seção abordaremos algumas dificuldades encontradas com o intuito de identificar novos trabalho e possíveis melhorias para trabalhos afins.

Inicialmente, todo novo paradigma de programação envolve uma nova maneira de pensar no que diz respeito a modelagem do problema em questão. O formalismo Gamma fornece uma maneira muito simples e direta de expressar problemas. Realmente a facilidade em expressar problemas é algo bastante relevante e, por si só, motivo de investir em pesquisas neste modelo computacional com alto potencial para execução em diversas plataformas paralelas de hardware. Porém, inicialmente, o programador precisa adaptar-se a este novo paradigma para expressar problemas. O fato relatado em si não é considerado uma dificuldade encontrada, porém deve ser levado em consideração mediante a decisão de investimentos de pesquisas na área.

Um aspecto que impactou diretamente na adoção de um grafo completo para a solução proposta foi a impossibilidade de estipulação de uma condição de parada para uma desejável reação que iria inserir em nosso multiconjunto tuplas do tipo

aresta, conforme apresentamos na Subseção 4.3.2.1. Porém, esta dificuldade diz respeito à especificidade da solução proposta e não diz respeito diretamente ao modelo computacional a que a solução foi submetida.

Um ponto importante a ser mencionado que diz respeito, não ao formalismo Gamma, mas sim à implementação de Gamma utilizada é falta de suporte a algumas operações matemáticas necessárias a esta solução, como por exemplo a radiciação. Além disso, a implementação só fornecia suporte a números inteiros. Dessa forma, em possíveis extensões e aprimoramentos a esta implementação sugere-se suporte a números reais de ponto flutuante além da possibilidade de utilização de outras funções matemáticas.

Uma sugestão de funcionalidade adicional poderia ser a utilização de mais de um multiconjunto. Este aspecto poderia de alguma forma facilitar o trabalho do programador uma vez que durante o desenvolvimento deste trabalho sentiu-se a necessidade de utilização de mais de um multiconjunto, como por exemplo no primeiro estágio desta solução os dados provenientes de cada sensor poderiam estar dispostos em um multiconjunto específico.

Por fim uma dificuldade encontrada neste trabalho foi a inexistência de documentação acerca da sintaxe suportada pela implementação específica de Gamma utilizada. Assim, para descobrir as palavras reservadas, operadores, funções matemáticas entre outros símbolos suportados, tivemos que verificar detalhes de implementação do compilador Gamma utilizado. Diante disso, na Tabela 4.1 foi descrita toda a simbologia suportada por esta implementação de Gamma para futuras utilizações.

4.4 Trabalhos Correlatos

Na busca por trabalhos correlatos, não foi encontrada nenhuma representação de um problema real na área de fusão de dados que utilizou o formalismo Gamma nem tampouco a implementação do problema de fusão de dados sobre alguma solução paralela. Dessa forma, iremos abordar nesta seção algumas aplicações em diversas outras áreas de grande interesse.

Na área de processamento de imagens, podemos citar os trabalhos de CHRISTIAN CREVEUIL [41] e BANÂTRE e LE MÉTAYER [9]. No primeiro deles, [41], Gamma foi utilizado em uma aplicação de processamento de imagens que tem por objetivo o reconhecimento da topografia tridimensional de uma rede vascular cerebral contida em duas radiografias. A utilização de Gamma, comparada a solução anterior em outra linguagem, trouxe benefícios como melhor entendimento do ponto chave do problema além de reduzir significativamente a quantidade de erros. Já em [9], utilizou-se Gamma para detecção de bordas de objetos em imagens representa-

das em escala de cinza. A solução calculava o gradiente para cada pixel em relação a seus vizinhos e após a verificação da intensidade do gradiente era determinado se o pixel fazia ou não parte da borda do objeto. Conforme representamos em nossa solução, aqui os objetos no multiconjunto eram representados por tuplas.

Em MURTHY e KRISHNAMURTHY [39] e [9], são fornecidas aplicações para cálculo do caminho mais curto (*shortest path*) em um grafo dirigido. Conforme implementamos em nossa solução, aqui cada aresta possui um custo associado e o cálculo do caminho mais curto (melhor caminho) é realizado através do somatório dos custos das arestas que compõem um caminho. Dessa forma, o caminho com menor somatório de custo é considerado o melhor caminho.

Ainda em [9], são apresentados uma série de programas para outros domínios de aplicação, como por exemplo, problemas para processamento de *strings*, problemas geométricos, problemas de ordenação, problemas de sincronização de processos, entre outros. Com relação aos problemas geométricos, a aplicação existente encontra um fecho convexo de um conjunto de pontos de um plano, o qual é definido como o menor polígono convexo de um conjunto contendo todos estes pontos. Na categoria de sincronização de processos, foi abordado o problema clássico do jantar dos filósofos porém com foco nos estados observáveis da computação e não exclusivamente no resultado.

Capítulo 5

Validação da Solução

No capítulo anterior, 4, foram apresentadas as características da solução proposta neste trabalho. O presente capítulo tem por objetivo apresentar os experimentos desenvolvidos além de expor os resultados obtidos e a discussão destes resultados.

Conforme veremos na Seção 5.4.1, após o desenvolvimento de uma abordagem iterativa, onde o primeiro e o segundo estágio do algoritmo foram interligados por um código desenvolvido em linguagem C, três abordagens distintas foram propostas (*Sequencial*, *Paralela* e *Híbrida*), com o intuito de analisar formas de explorar o paralelismo em Gamma. Vale ressaltar que a abordagem *Sequencial* não explora nenhum tipo de paralelismo, porém, para fins de análise das demais abordagens, se faz necessária uma abordagem base para comparação. Dessa maneira, surge a abordagem *Sequencial*, que reflete o pior caso, uma vez que não existe nenhum tipo de paralelismo a ser executado entre as reações.

Como vimos no parágrafo anterior, o algoritmo proposto é iterativo e cíclico, uma vez que os estágios foram interligados por um código em linguagem C e este mesmo código fornece novos plots para iterações subsequentes. Entretanto, para realizar uma análise onde se deseja mensurar o tempo de execução de um experimento é necessário selecionar parte do algoritmo o qual iremos medir o tempo desejado. Ao mesmo tempo, a parte deste algoritmo onde existe a possibilidade de ocorrência de paralelismo é o primeiro estágio, onde o processamento dos grafos criados para cada conjunto de dados de um mesmo sensor poderá executar em paralelo ao processamento de grafos pertencentes a outros sensores. Além disso, o segundo estágio do algoritmo proposto é idêntico, independente da abordagem utilizada, uma vez que este utiliza os melhores pontos de cada sensor e os processa em um único grafo, que contém os melhores plots de cada sensor. Portanto, para fins de verificação do tempo de execução destas abordagens (*Sequencial*, *Paralela* e *Híbrida*) e possibilitar tal execução em um ambiente remoto que permita o uso de algum hardware paralelo, tais abordagens executaram o primeiro estágio do algoritmo proposto. Assim, os experimentos que serão apresentados tem o objetivo de analisar formas de

explorar o paralelismo do algoritmo implementado além de realizar uma análise de desempenho das implementações de Gamma utilizadas.

5.1 Ambiente de Testes

De acordo com o que apresentamos na Subseção 4.1.1, foram utilizadas três implementações de Gamma (*Gamma-Sequencial*, *Gamma-MPI* e *Gamma-GPU*) sob as quais os casos de teste que serão descritos a seguir na Seção 5.2 foram executados. A implementação *Gamma-Sequencial* utiliza somente um processador para a execução de aplicações em Gamma, ao passo que as duas últimas implementações necessitam de ambiente que possua hardware paralelo composto de múltiplos processadores para execução de seus experimentos. Entretanto, a última implementação, *Gamma-GPU*, necessita adicionalmente de GPUs disponíveis. Dessa maneira, os experimentos que serão descritos neste capítulo foram executados em um ambiente distribuído com suporte a GPU.

Assim, para a realização deste trabalho, foi necessário o prestimoso apoio do CENAPAD-UFC (Centro Nacional de Processamento de Alto Desempenho da Universidade Federal do Ceará). O CENAPAD-UFC integra o SINAPAD (Sistema Nacional de Processamento de Alto Desempenho), que é composto por nove centros espalhados por todo o Brasil com intuito de fornecer processamento de alto desempenho com vistas para o desenvolvimento científico e tecnológico de nosso país.

O supercomputador utilizado possui arquitetura de cluster composto por vários servidores em *blade*. Seu processamento teórico é da ordem de 14,2 TeraFlops, conforme dados do próprio CENAPAD-UFC. Para tanto, seu hardware apresenta a seguinte configuração:

- 48 *Blades Bull* 500, onde cada uma é composta de: 2x processadores *Intel*® *Westmere*® X5650 EP; 6x 4GB DDR3 133MHz de memória RAM; e 1x 250GB SATA II 1,8" HDD, o que totaliza 576 núcleos;

- 3 *Blades Bullx* B505, onde cada uma é composta por: 1x processador *Intel*® *Xeon*® CPU E5-2470; 96GB DDR3 133MHz de Memória RAM; e 1x GPU NVidia *Tesla* K20m 5GB DDR5 de memória, totalizando 48 núcleos.

Quanto aos recursos de software, podemos citar o sistema operacional *Linux Red Hat* EL 6.4; Ambientes para compilação e execução em MPI e CUDA (Compilador nvcc); e SLURM (*Simple Linux Utility for Resource Management*). Este último consiste em um ambiente que gerencia a submissão e execução de *jobs*.

5.2 Casos de Teste

Os experimentos, ou casos de teste, utilizados neste trabalho podem ser agrupados em dois conjuntos. O primeiro (Paralelismo da Solução Proposta) visa analisar diferentes maneiras de explorar o paralelismo da solução proposta, mediante as ferramentas disponíveis em Gamma, sejam elas operadores de composição paralela (inter reação) ou exploração do paralelismo inerente a cada reação (intra reação). Já o segundo conjunto de experimentos (Desempenho das Implementações), visa analisar as implementações de Gamma utilizadas (*Gamma-Sequencial*, *Gamma-MPI* e *Gamma-GPU*) do ponto de vista de desempenho.

Para cada experimento proposto foi utilizada uma metodologia onde realizou-se dez execuções para cada experimento. Esta quantidade de execuções está intimamente relacionada ao desvio padrão que se desejava obter que, no pior caso, não ultrapassou 8% da média do tempo total de execução e, na grande maioria dos casos, este desvio padrão esteve abaixo dos 5% do tempo de execução do experimento.

Para o primeiro conjunto de experimentos (Paralelismo da Solução Proposta), o multiconjunto foi composto pela simulação de dados adquiridos por três sensores distintos para as três abordagens propostas (*Sequencial*, *Paralela* e *Híbrida*), sendo que a quantidade de plots por sensor foi igual a dez. Assim, para este experimento, pela utilização de um grafo completo (Subseção 4.3.1) cada sensor totaliza cerca de 100 elementos, entre vértices e arestas e o multiconjunto completo mais de 300 elementos. Portanto, neste conjunto de experimentos deseja-se verificar qual das três abordagens propostas será mais adequada para a realização do segundo conjunto de testes.

Tabela 5.1: Paralelismo da solução proposta - experimentos

Experimentos		
Abordagem	**Configuração**	**Tamanho da Entrada**
Sequencial	3 sensores 10 Plots por sensor	300
Paralela	3 sensores 10 Plots por sensor	300
Híbrida	3 sensores 10 Plots por sensor	300

Para o segundo grupo de experimentos, visando analisar o desempenho das implementações propostas, foram elaboradas diversas entradas somente para a abordagem chamada *Híbrida*, uma vez que esta foi a abordagem que apresentou melhor desempenho no primeiro caso de teste (Paralelismo da Solução Proposta). As entradas utilizadas para o segundo grupo de experimentos variam tanto a quantidade de sensores envolvidos (3, 6, 9 ou 11) quanto a quantidade de plots por

sensor (10, 20 ou 30). Assim, na abordagem com menor quantidade de elementos (3 sensores, 10 plots por sensor) o multiconjunto é formado por cerca de 300 elementos. Já na abordagem com maior quantidade de elementos (11 sensores 30 elementos) o multiconjunto possui cerca de 9900 elementos.

Tabela 5.2: Desempenho das implementações - experimentos

Experimentos		
Abordagem	**Configuração**	**Tamanho da Entrada**
Híbrida	3 sensores 10 Plots por sensor	300
Híbrida	3 sensores 20 Plots por sensor	1200
Híbrida	3 sensores 30 Plots por sensor	2700
Híbrida	6 sensores 10 Plots por sensor	600
Híbrida	6 sensores 20 Plots por sensor	2400
Híbrida	6 sensores 30 Plots por sensor	5400
Híbrida	9 sensores 10 Plots por sensor	900
Híbrida	9 sensores 20 Plots por sensor	3600
Híbrida	9 sensores 30 Plots por sensor	8100
Híbrida	11 sensores 10 Plots por sensor	1100
Híbrida	11 sensores 20 Plots por sensor	4400
Híbrida	11 sensores 30 Plots por sensor	9900

Na Seção 5.3, apresentaremos os resultados para todos os experimentos mencionados.

5.3 Resultados Obtidos

Nesta seção iremos expor os resultados para os casos de testes voltados a analisar aspectos inerentes ao paralelismo da solução proposta nas três abordagens desenvolvidas: *Sequencial*, *Paralela* e *Híbrida*. Apesar do foco deste trabalho não ser a análise do desempenho das implementações de Gamma utilizadas, a solução desenvolvida mostra-se como uma interessante base de testes das três implementações usadas, *Gamma-Sequencial*, *Gamma-MPI* e *Gamma-GPU*. Assim, nas Subseções 5.3.1 e 5.3.2 serão apresentados os resultados que serão posteriormente discutidos na Seção 5.4.

5.3.1 Paralelismo da Solução

Veremos agora os resultados obtidos para os casos de teste que visaram analisar o paralelismo da solução proposta, tendo em vista as três abordagens desenvolvidas.

Tabela 5.3: Paralelismo da solução - tempos de execução - abordagens Sequencial, Paralela e Híbrida para as implementações Gamma-Sequencial, Gamma-MPI e Gamma-GPU - 3 sensores, 10 plots por sensor.

		Implementações de Gamma					
		Gamma-Sequencial		Gamma-MPI		Gamma-GPU	
		Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Abordagem	Sequencial	0,128	0,004	11,716	0,090	55,536	1,732
	Paralela	0,126	0,005	8,082	0,045	44,902	1,369
	Híbrida	0,122	0,004	4,782	0,316	15,178	0,659

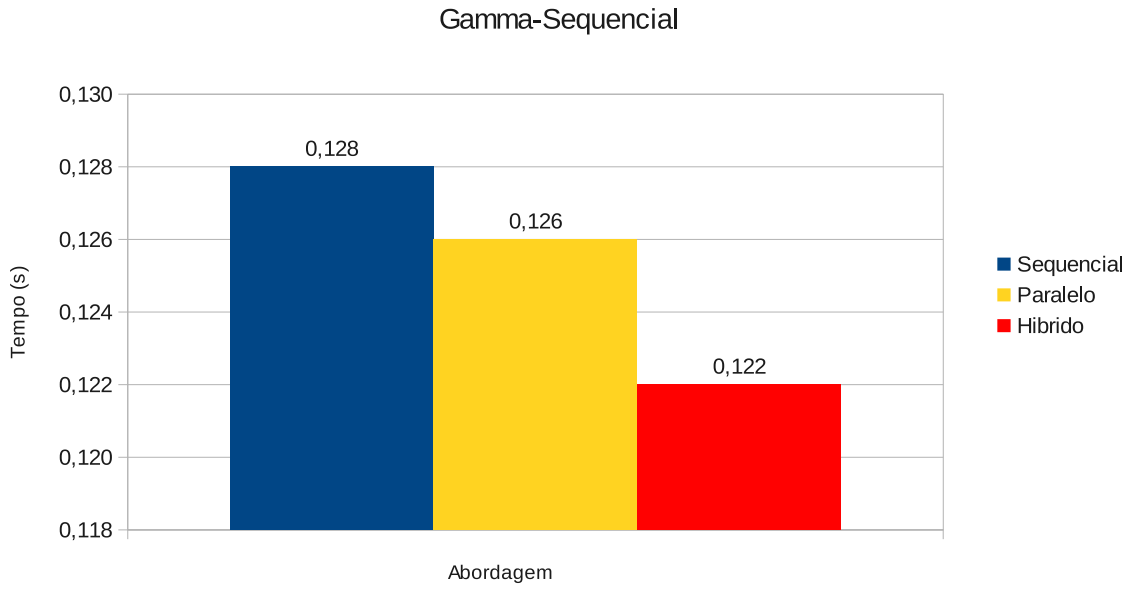

Figura 5.1: Tempos de execução - abordagens Sequencial, Paralela e Híbrida utilizando a implementação Gamma-Sequencial - 3 sensores, 10 plots por sensor

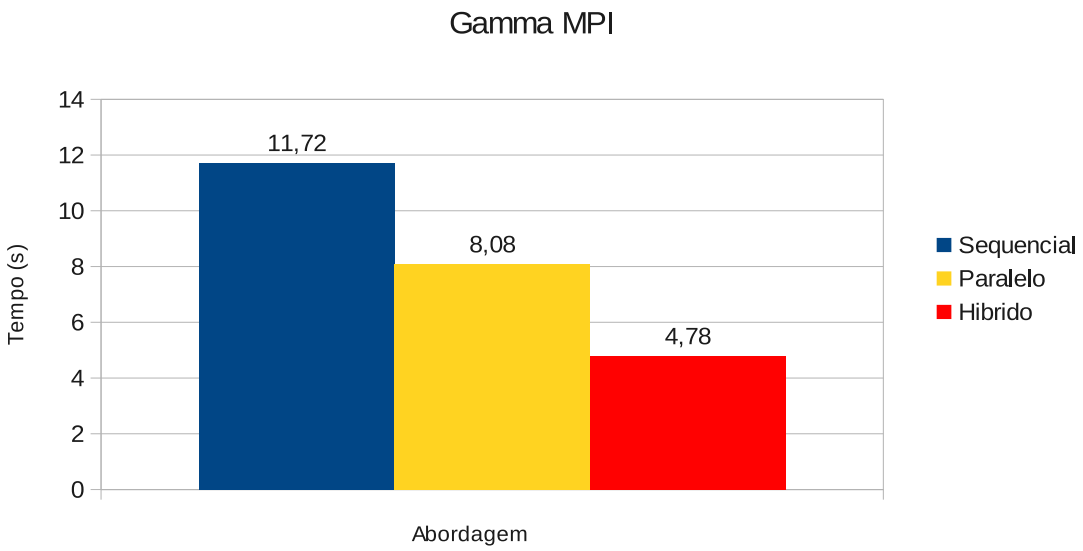

Figura 5.2: Tempos de execução - abordagens Sequencial, Paralela e Híbrida utilizando a implementação Gamma-MPI - 3 sensores, 10 plots por sensor

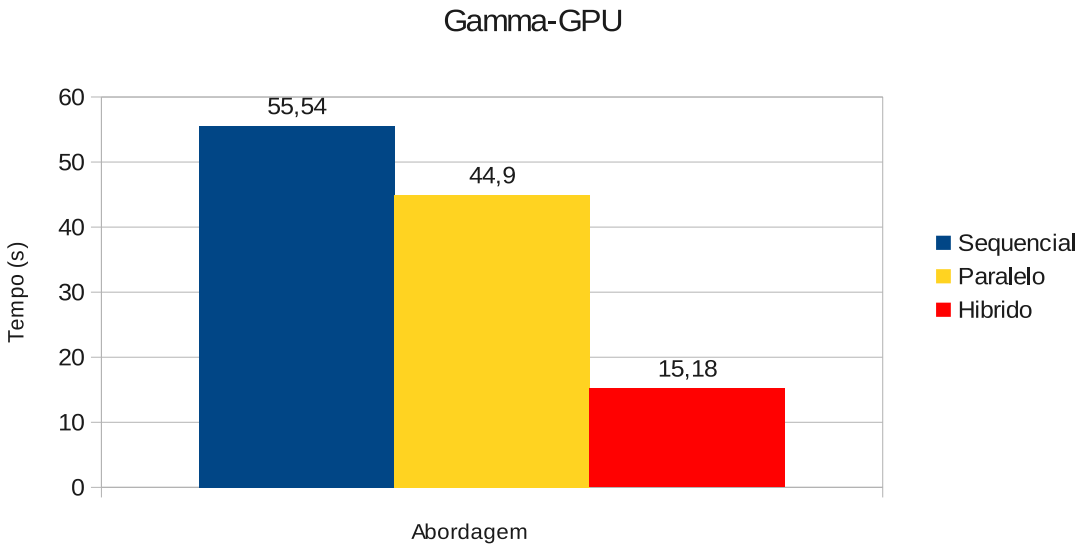

Figura 5.3: Tempos de execução - abordagens Sequencial, Paralela e Híbrida utilizando a implementação Gamma-GPU - 3 sensores, 10 plots por sensor

5.3.2 Desempenho das Implementações de Gamma

A seguir, serão abordados os resultados obtidos para os casos de teste que visaram analisar as três implementações de Gamma utilizadas.

5.3.2.1 3 sensores

Tabela 5.4: Tempos de execução - abordagem Híbrida - implementações Gamma-Sequencial, Gamma-MPI e Gamma-GPU - 3 sensores - 10, 20 e 30 plots por sensor.

3 Sensores		Entradas					
		3se10p		3se20p		3se30p	
		Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Implementações	Gamma-Sequencial	0,12	0	3,732	0,004	28,242	0,022
	Gamma-MPI	4,97	0,22	15	1,035	68,706	0,557
	Gamma-GPU	15,392	0,333	29,688	0,462	96,036	1,466

Tabela 5.5: Speedup - abordagem Híbrida - Gamma-MPI e Gamma-GPU comparados ao Gamma-Sequencial - 3 sensores - 10, 20 e 30 plots por sensor.

3 Sensores		Entradas		
		3se10p	3se20p	3se30p
		Speedup	Speedup	Speedup
Implementações	Gamma-MPI	0,024	0,248	0,411
	Gamma-GPU	0,007	0,125	0,294

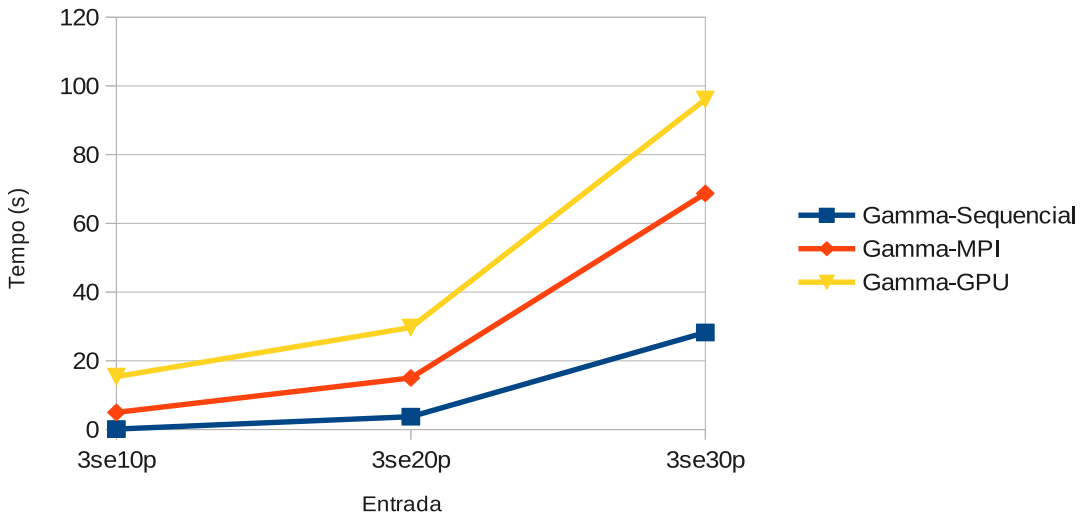

Figura 5.4: Tempos de execução - abordagem Híbrida - 3 sensores (10, 20 e 30 plots por sensor)

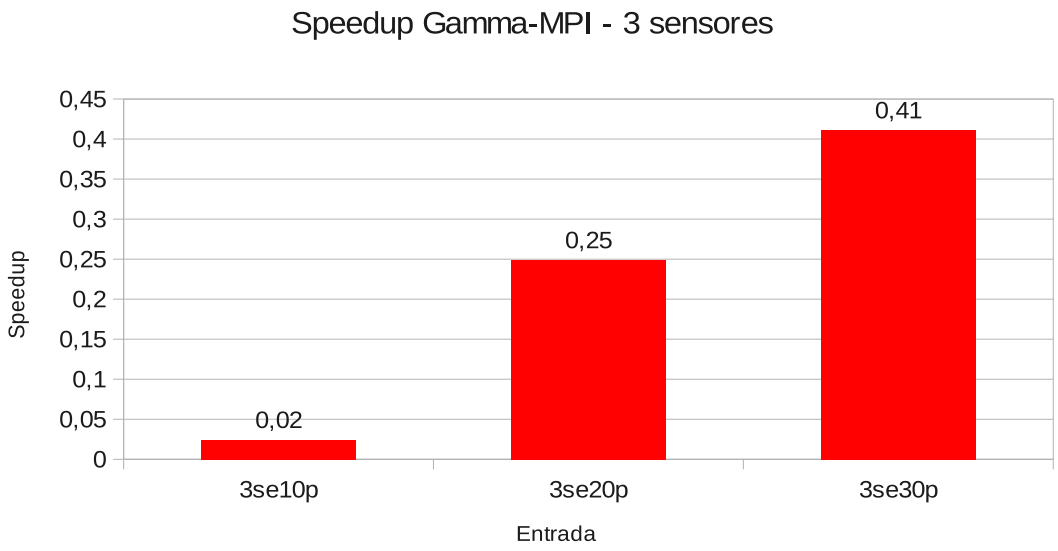

Figura 5.5: Gráfico de speedup - implementação Gamma-MPI - abordagem Híbrida - 3 sensores (10, 20 e 30 plots por sensor)

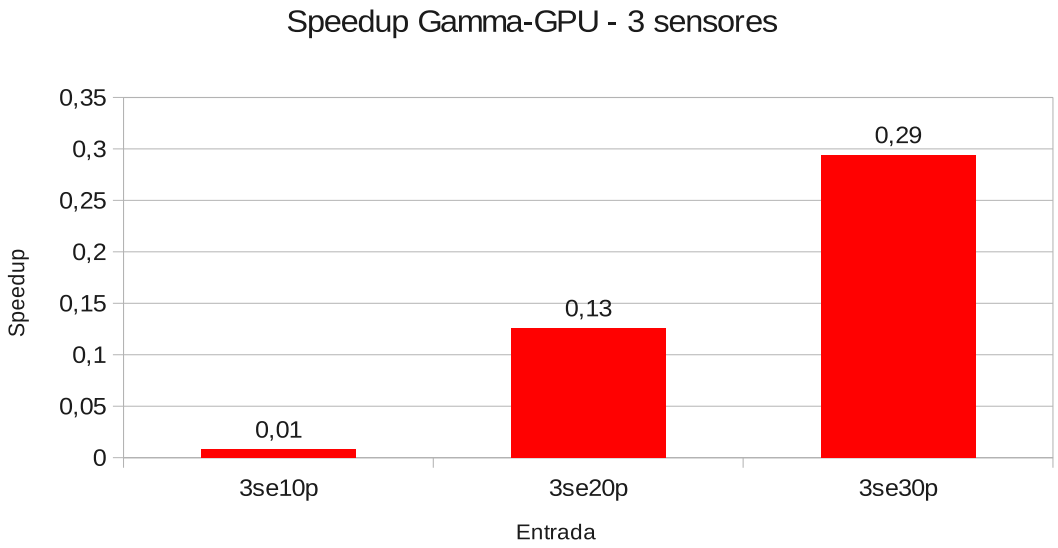

Figura 5.6: Gráfico de speedup - implementação Gamma-GPU - abordagem Híbrida - 3 sensores (10, 20 e 30 plots por sensor)

5.3.2.2 6 sensores

Tabela 5.6: Tempos de Execução - abordagem Híbrida - implementações Gamma-Sequencial, Gamma-MPI e Gamma-GPU - 6 sensores - 10, 20 e 30 plots por sensor.

6 Sensores		Entradas					
		6se10p		6se20p		6se30p	
		Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Implementações	Gamma-Sequencial	1,146	0,005	34,812	0,021	274,232	1,681
	Gamma-MPI	7,11	0,313	87,286	1,208	535,134	18,671
	Gamma-GPU	21,448	0,247	95,258	1,186	316,256	9,664

Tabela 5.7: Speedup - abordagem Híbrida - Gamma-MPI e Gamma-GPU comparados ao Gamma-Sequencial - 6 sensores - 10, 20 e 30 plots por sensor.

6 Sensores		Entradas		
		6se10p	6se20p	6se30p
		Speedup	Speedup	Speedup
Implementações	Gamma-MPI	0,161	0,398	0,512
	Gamma-GPU	0,053	0,365	0,867

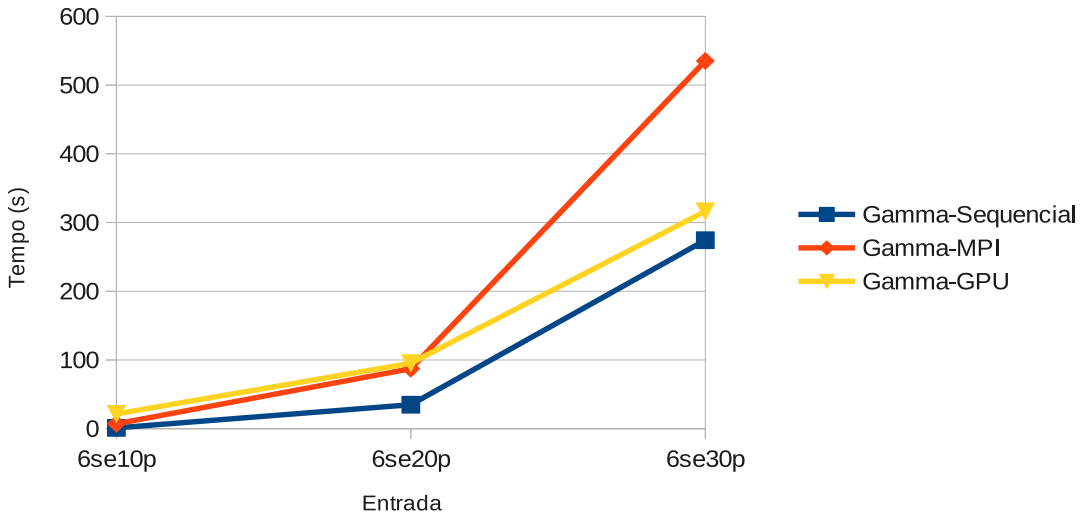

Figura 5.7: Tempos de execução - abordagem Híbrida - 6 sensores (10, 20 e 30 plots por sensor)

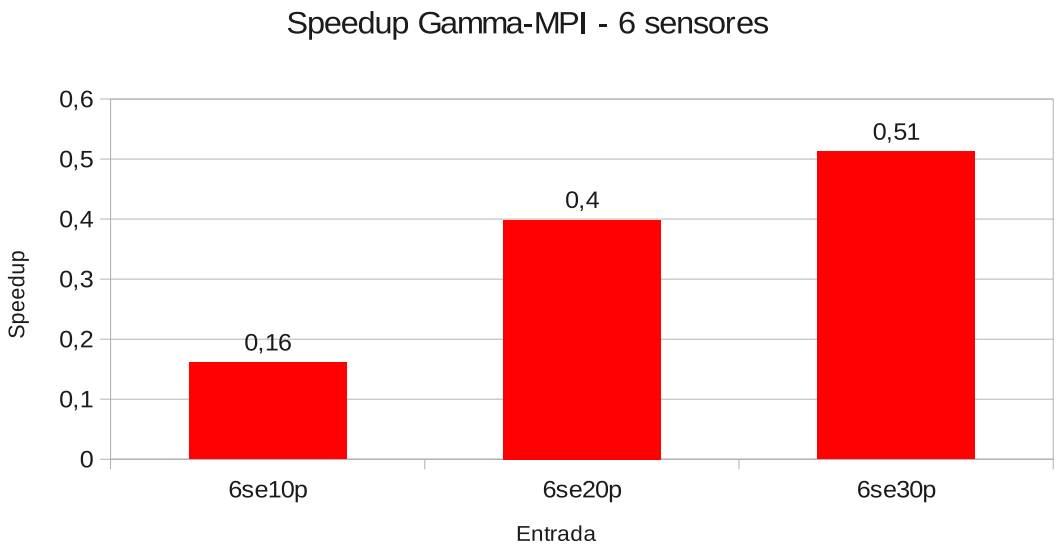

Figura 5.8: Gráfico de speedup - implementação Gamma-MPI - abordagem Híbrida - 6 sensores (10, 20 e 30 plots por sensor)

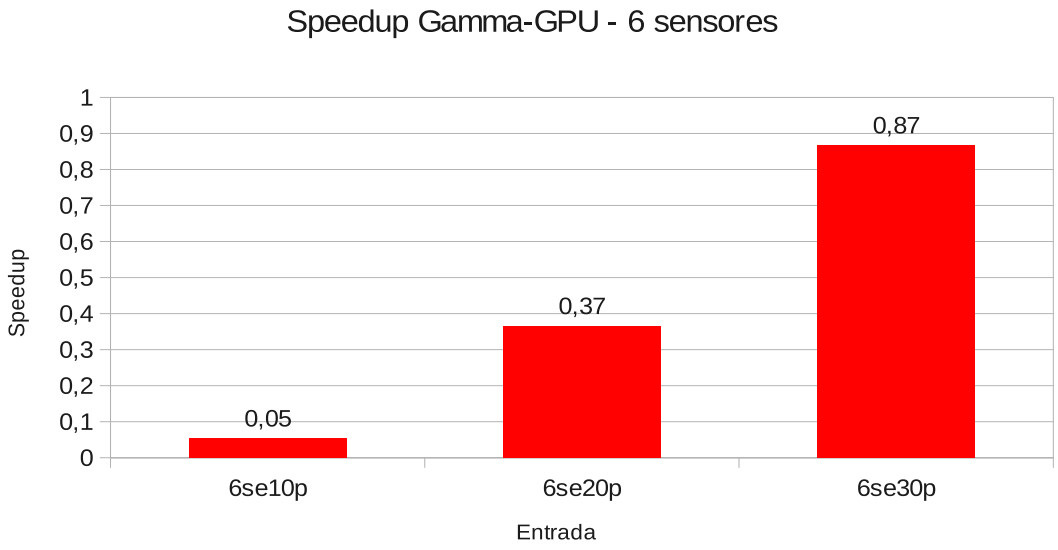

Figura 5.9: Gráfico de speedup - implementação Gamma-GPU - abordagem Híbrida - 6 sensores (10, 20 e 30 plots por sensor)

5.3.2.3 9 sensores

Tabela 5.8: Tempos de execução - abordagem Híbrida - implementações Gamma-Sequencial, Gamma-MPI e Gamma-GPU - 9 sensores - 10, 20 e 30 plots por sensor.

9 Sensores		Entradas					
		9se10p		9se20p		9se30p	
		Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Implementações	Gamma-Sequencial	4,426	0,005	141,79	1,240	1227,86	1,732
	Gamma-MPI	14,528	0,679	272,34	4,628	2009,802	19,883
	Gamma-GPU	32,056	0,672	195,066	0,179	814,694	26,957

Tabela 5.9: Speedup - abordagem Híbrida - Gamma-MPI e Gamma-GPU comparados ao Gamma-Sequencial - 9 sensores - 10, 20 e 30 plots por sensor.

9 Sensores		Entradas		
		9se10p	9se20p	9se30p
		Speedup	Speedup	Speedup
Implementações	Gamma-MPI	0,304	0,520	0,610
	Gamma-GPU	0,138	0,726	1,507

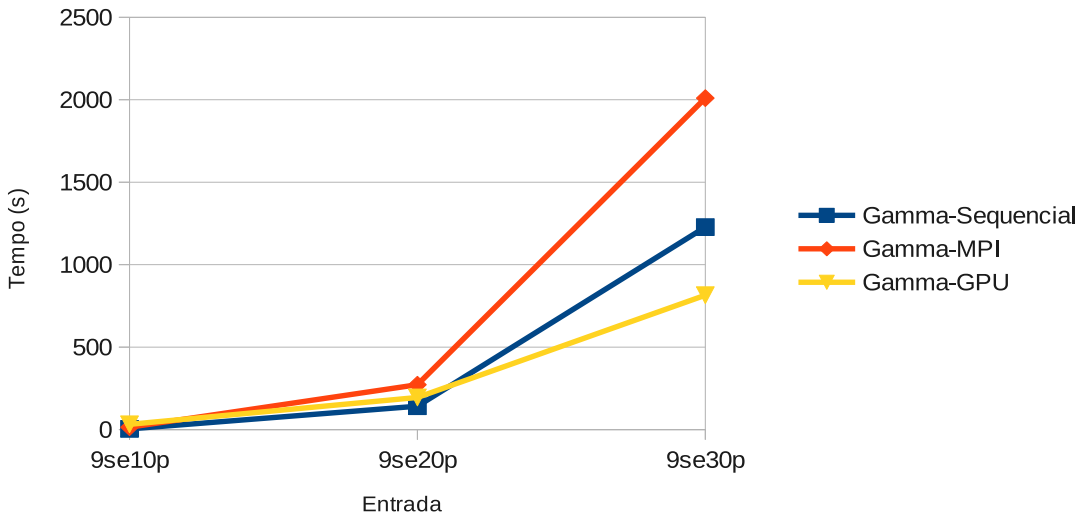

Figura 5.10: Tempos de execução - abordagem Híbrida - 9 sensores (10, 20 e 30 plots por sensor)

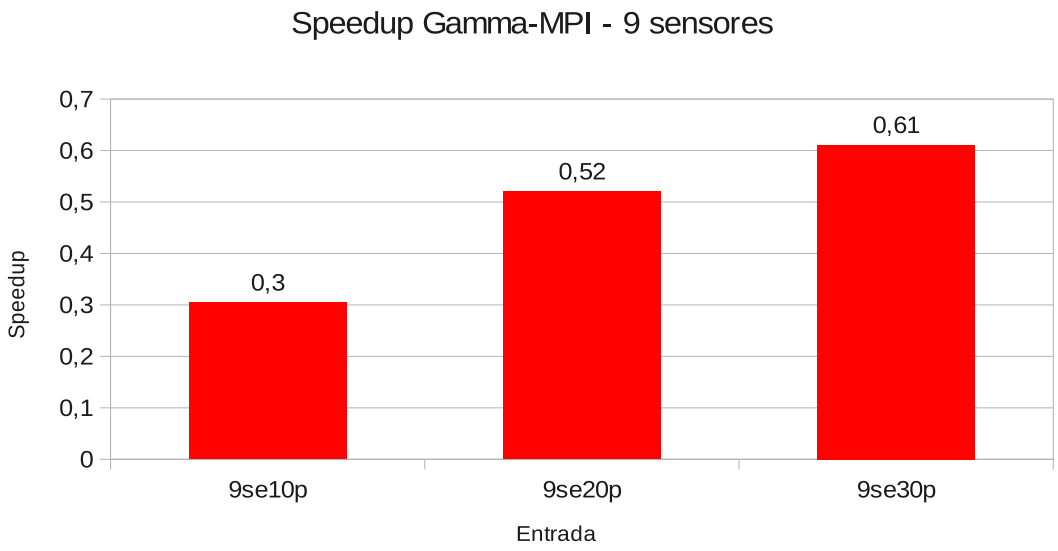

Figura 5.11: Gráfico de speedup - implementação Gamma-MPI - abordagem Híbrida - 9 sensores (10, 20 e 30 plots por sensor)

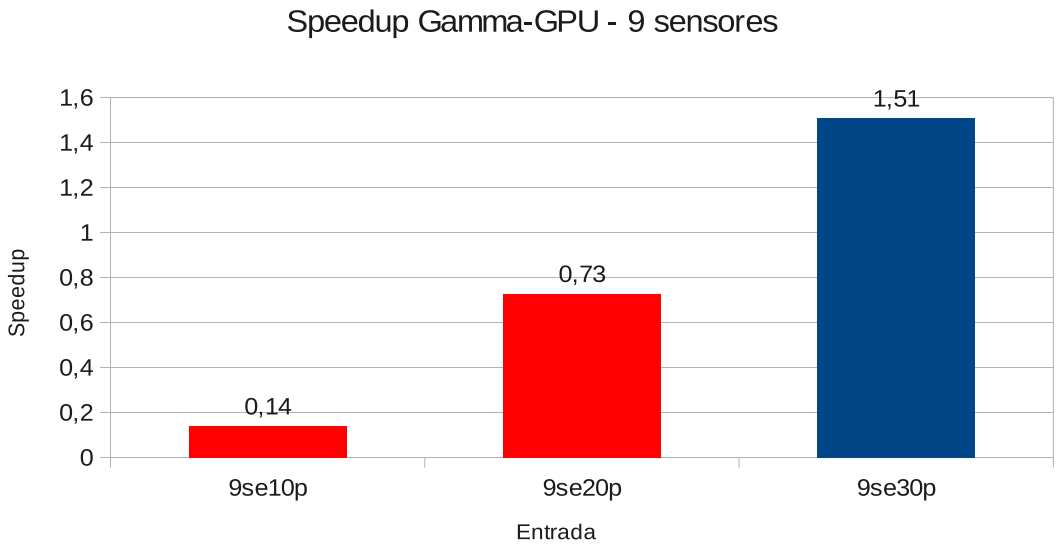

Figura 5.12: Gráfico de speedup - implementação Gamma-GPU - abordagem Híbrida - 9 sensores (10, 20 e 30 plots por sensor)

5.3.2.4 11 sensores

Tabela 5.10: Tempos de execução - abordagem Híbrida - implementações Gamma-Sequencial, Gamma-MPI e Gamma-GPU - 11 sensores - 10, 20 e 30 plots por sensor.

11 Sensores		Entradas					
		11se10p		11se20p		11se30p	
		Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
Implementações	Gamma-Sequencial	8,818	0,008	282,31	1,420	3350,38	18,490
	Gamma-MPI	25,758	0,227	534,072	14,232	4262,564	79,335
	Gamma-GPU	46,76	0,575	269,402	2,239	1858,238	34,302

Tabela 5.11: Speedup - abordagem Híbrida - Gamma-MPI e Gamma-GPU comparados ao Gamma-Sequencial - 11 sensores - 10, 20 e 30 plots por sensor.

11 Sensores		Entradas		
		11se10p	11se20p	11se30p
		Speedup	Speedup	Speedup
Implementações	Gamma-MPI	0,342	0,528	0,786
	Gamma-GPU	0,188	1,047	1,802

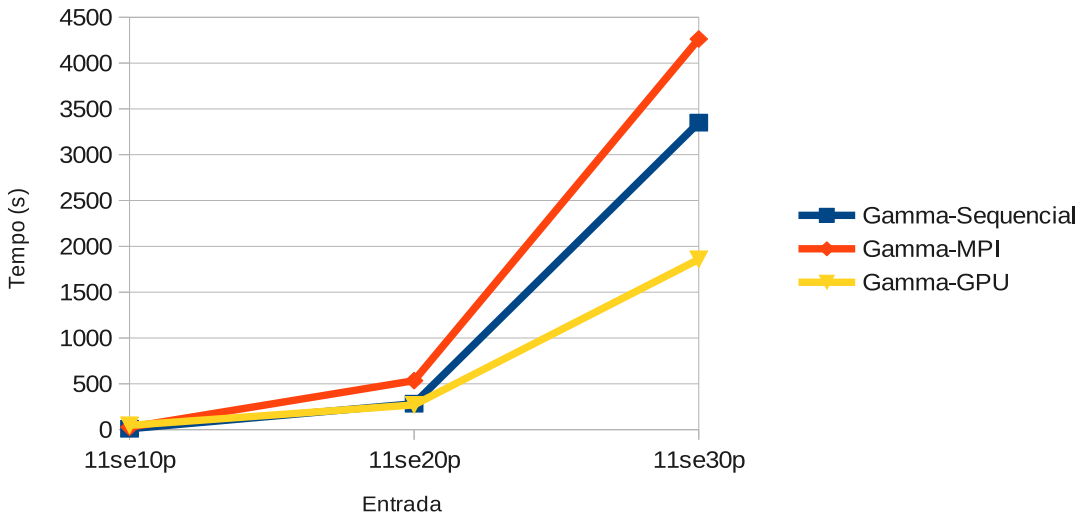

Figura 5.13: Tempos de execução - abordagem Híbrida - 11 sensores (10, 20 e 30 plots por sensor)

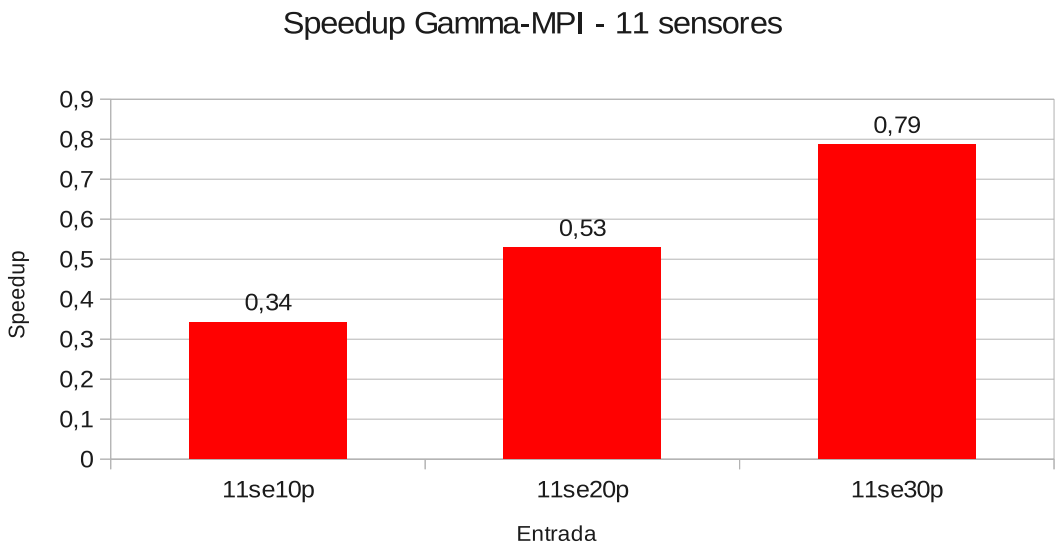

Figura 5.14: Gráfico de speedup - implementação Gamma-MPI - abordagem Híbrida - 11 sensores (10, 20 e 30 plots por sensor)

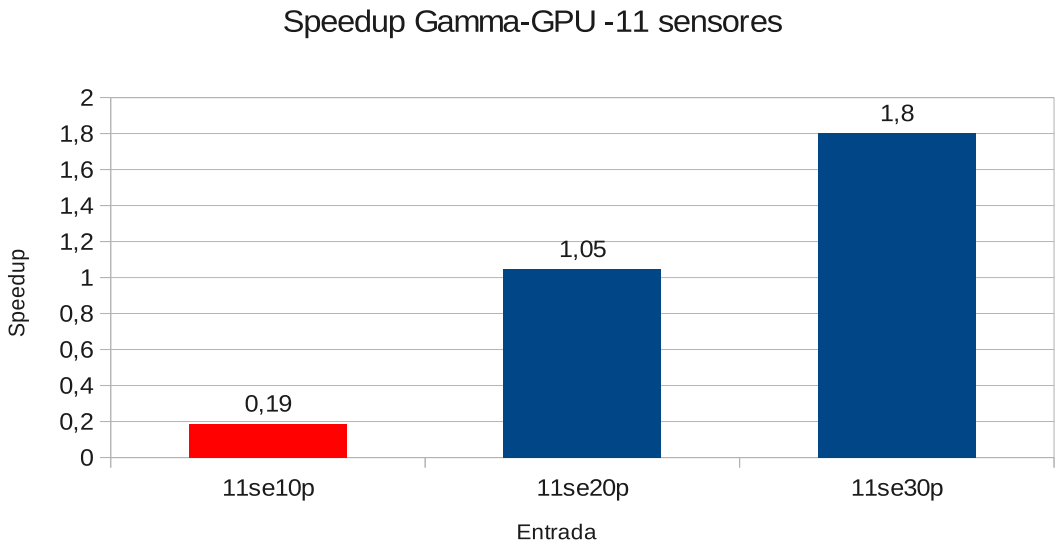

Figura 5.15: Gráfico de speedup - implementação Gamma-GPU - abordagem Híbrida - 11 sensores (10, 20 e 30 plots por sensor)

5.4 Discussão dos Resultados

Tendo exposto os resultados de todos os experimentos realizados através dos gráficos e tabelas acima, nesta seção iremos tecer alguns comentários sobre estes resultados tanto para os experimentos que visaram analisar características relacionadas à possibilidade de paralelismo quanto aos testes que visaram analisar o desempenho das três implementações utilizadas como base para nossos casos de teste.

5.4.1 Considerações sobre paralelismo da solução

Na Tabela 5.3 e nas Figuras 5.1, 5.2 e 5.3 foram expostos os resultados obtidos para os casos de teste categorizados como sendo inerentes a análise do paralelismo da solução proposta. Conforme podemos verificar, em todas as três implementações de Gamma utilizadas (*Gamma-Sequencial* fig. 5.1, *Gamma-MPI* fig. 5.2 e *Gamma-GPU* fig. 5.3) a abordagem *Híbrida* teve melhores resultados comparada à abordagem *Sequencial* e à abordagem *Paralela*. Porém, na implementação *Gamma-GPU* a abordagem *Híbrida* teve maiores melhorias do que as verificadas nas demais implementações, o que demonstra grande potencial de utilização da abordagem Híbrida associada a esta implementação de Gamma. É importante notar na Tabela 5.3 que os resultados de todos os experimentos realizados sobre a implementação *Gamma-Sequencial* foram melhores que os obtidos em outras implementações. Este fato está relacionado ao seguintes aspectos:

- Pelo tamanho da entrada utilizada (multiconjunto formado por cerca de 300 elementos), o *overhead* de comunicação necessário às implementações *Gamma-MPI* e *Gamma-GPU* não compensou a utilização destes dois modelos. Em outras palavras, tanto a versão *Gamma-MPI* quanto a versão *Gamma-GPU* utilizam troca de mensagens através de *Message Passing Interface* (MPI). Desta forma, ao comparar tais implementações com a implementação *Gamma-Sequencial*, essas possuem computação adicional específicas para toda a gerência de troca de mensagens necessária. Assim, se no cálculo da computação total necessária o custo computacional desta gerência de troca de mensagens não for pequeno ao ponto de não impactar o tempo total de execução, a escolha pode não ser vantajosa;

- A gerência de execução das reações e distribuição do multiconjunto nas implementações utilizadas centraliza o controle em uma única célula de processamento. Esta centralização gera um gargalo na computação que poderia ser bastante reduzido com a implementação da *"Proposta de um Escalonador para Gamma"*, trabalho este sugerido por Felipe França e Gabriel Paillard em [42].

Por fim é importante percebermos que, conforme já mencionado, a abordagem *Híbrida* apresentou melhores desempenhos que as demais em todas as implementações de Gamma utilizadas. Analisando as duas abordagens de maneira teórica, os resultados obtidos pelas abordagens *Paralela* e *Híbrida* deveriam ser bastante semelhantes, uma vez que na abordagem *Paralela* a cada instante de tempo uma reação de cada conjunto de sensor deveria estar sendo executada. Porém, conforme vimos acima, a gerência de execução de reações e distribuição centralizada do multiconjunto utilizada pelas implementações paralelas de Gamma (*Gamma-MPI* e *Gamma-GPU*) gera um gargalo pela utilização de uma única célula gerenciadora do processamento. A abordagem *Híbrida* sofre menos com esta característica da implementação, uma vez que a quantidade de reações envolvida é menor e a exploração do paralelismo é realizada de maneira intra reação, conforme vimos na Subseção . Por estes motivos, a abordagem *Paralela* apresenta resultados piores comparada à abordagem *Híbrida*.

Concluindo, vale a pena mencionar que pode causar dúvida se a implementação *Gamma-Sequencial* consegue executar e portanto medir o tempo de execução das abordagens *Paralela* e *Híbrida*, conforme apresentado no gráfico 5.1. Porém, é importante lembrar que apesar da implementação *Gamma-Sequencial* utilizar somente um processador, esta implementação fornece um pseudo paralelismo ao programador, pela alternância de execuções das reações neste único processador. Em outras palavras, os recursos utilizados pelas abordagens *Paralela* e *Híbrida* para expressarem o paralelismo (operador "|"e comparação de elementos advindos de um mesmo sensor, vide Seção 5.4.1) são perfeitamente suportados pela implementação *Gamma-Sequencial*.

5.4.2 Análise de Desempenho das Implementações

A presente seção destina-se a analisar o desempenho das três implementações de Gamma utilizadas, sejam elas *Gamma-Sequencial*, *Gamma-MPI* e *Gamma-GPU*. Maiores informações a respeito destas implementações podem ser verificadas na Seção 4.1.1.

Tendo em vista os resultados obtidos na seção anterior, para a análise das implementações mencionadas iremos utilizar a abordagem *Híbrida*. Conforme os resultados apresentados nos gráficos e tabelas da Subseção 5.3.2, foram realizados experimentos simulando a utilização de dados advindos de uma quantidade de sensores que varia entre 3, 6, 9 e 11. Além disso, a quantidade de plots por sensor, para cada quantidade de sensores descrita acima, variou entre 10, 20 e 30. Dessa maneira, os casos de testes foram agrupados por quantidade de sensores, ou seja, os gráficos e tabelas foram construídos para dados advindos de 3 sensores (3 sensores

variando a quantidade de plots em 10, 20 e 30 - Tabelas 5.4, 5.5 e gráficos 5.4, 5.5 e 5.6), 6 sensores (6 sensores variando a quantidade de plots em 10, 20 e 30 - Tabelas 5.6, 5.7 e gráficos 5.7, 5.8 e 5.9), 9 sensores (9 sensores variando a quantidade de plots em 10, 20 e 30 - Tabelas 5.8, 5.9 e gráficos 5.10, 5.11 e 5.12 e 11 sensores (11 sensores variando a quantidade de plots em 10, 20 e 30 - Tabelas 5.10, 5.11 e gráficos 5.13, 5.14 e 5.15).

Além das informações de tempo de execução e desvio padrão, inseriu-se a informação de speedup em uma tabela adicional, onde esta foi calculada sempre em comparação com a implementação *Gamma-Sequencial*. O *Speedup* expressa o quão mais rápido uma execução é comparada a outra. Em outras palavras, foi calculado o speedup da implementação *Gamma-MPI* sobre a *Gamma-Sequencial* e posteriormente *Gamma-GPU* sobre *Gamma-Sequencial*. Assim:

$$Speedup_{(MPI)} = \frac{T_{Gamma-Sequencial}}{T_{Gamma-MPI}}$$

$$Speedup_{(GPU)} = \frac{T_{Gamma-Sequencial}}{T_{Gamma-GPU}}$$

Pelos dados analisados, podemos verificar que a medida em que o multiconjunto aumenta, a implementação *Gamma-GPU* vai aproximando seu desempenho à *Gamma-Sequencial*, chegando a obter melhores desempenhos nos casos de teste que utilizam 9 sensores e 30 plots por sensor, 11 sensores com 20 plots por sensor e 11 sensores com 30 plots por sensor. Outro fato que chama a atenção é a ineficiência de *Gamma-GPU* frente à *Gamma-MPI* em casos que utilizaram entradas consideradas pequenas (3 sensores). Isto pode ser justificado pelo fato de, para este tamanho de entrada, há pouco paralelismo disponível a ser explorado pelas GPUs, fazendo com que o potencial ganho de desempenho seja sobreposto pelo *overhead* inserido devido às cópias dos dados entre as memórias da CPU e GPU.

Para os resultados dos experimentos com 3 sensores (gráfico 5.4), podemos verificar que *Gamma-Sequencial* possui desempenho melhor que as duas outras implementações, chegando a apresentar um tempo médio de execução de cerca de 28 segundos contra 68 segundos do *Gamma-MPI* e 96 segundos do *Gamma-GPU* para a execução 3s30 (3 sensores, 30 plots por sensor). Por isso, os *Speedups* expressos nos gráficos 5.5 e 5.6 foram menores que "1" (o maior valor de *"speedup"* foi 0,41 para a execução 3s30 em *Gamma-MPI*), configurando assim um *Slowdown*, que indica que, neste caso, não houve ganho de desempenho comparado à implementação *Gamma-Sequencial*. Para fins deste trabalho, indicaremos pela cor vermelha as barras dos gráficos que apresentam *slowdown* e pela cor azul quando houver *speedup* (valor de *speedup* maior que 1).

Para o conjunto de resultados referente às execuções com 6 sensores, apesar

de ainda incorrer em *Slowdown* os resultados da implementação *Gamma-GPU* aproximaram-se de *Gamma-Sequencial* conseguindo tempos de execução de 316 segundos frente a 274 segundos obtidos na implementação *Gamma-Sequencial*. Assim, os valores de *"Speedup"* obtidos para este conjunto de execuções para 6 sensores chegaram a 0,86 para a entrada 6s30 para a implementação *Gamma-GPU* comparado a *Gamma-Sequencial*.

Já para os resultados dos casos de teste que utilizaram dados provenientes de 9 sensores, no gráfico 5.12, verificamos um *Speedup* de 1,507 para *Gamma-GPU*, onde seu tempo de execução foi de 814 segundos frente a 1227 segundos para *Gamma-Sequencial* e 2009 segundos para *Gamma-MPI*.

Por fim, os experimentos realizados para uma base de dados com 11 sensores, obtiveram os melhores resultados, onde verificamos dois *speedups* para as execuções na implementação *Gamma-GPU*, conforme podemos verificar na Tabela 5.11. O primeiro deles refere-se a execução composta por 11 sensores, 20 plots por sensor, onde o *speedup* obtido foi de 1,047 e o tempo de execução em *Gamma-GPU* foi de 269 segundos quando *Gamma-Sequencial* obteve 282 segundos e Gamma-MPI atingiu 534 segundos. Já o segundo *speedup* obtido, também pela implementação *Gamma-GPU* foi de 1,802 quando esta implementação obteve 1858 segundos de execução comparado a 3350 segundos (*Gamma-Sequencial*) e 4262 segundos (*Gamma-MPI*).

Estes resultados demonstram que, para o algoritmo de fusão de dados proposto, a abordagem *Gamma-GPU* é mais vantajosa para grandes entradas (9 sensores com 30 plots por sensor, 11 sensores contendo 20 plots para cada sensor e 11 sensores com 30 plots cada), pois provavelmente o custo-benefício de utilização das GPU está sendo positivo.

O fato de *Gamma-MPI* não ter obtido bons resultados está diretamente relacionado aos aspectos já mencionados na seção anterior, que são:

- o overhead de comunicação necessária à toda a gerência de troca de mensagens entre os processadores impacta no tempo de computação, principalmente em casos de pequenas entradas; e
- a gerência de execução das reações e distribuição do multiconjunto realizada de forma centralizada gera um gargalo computacional.

Assim, tanto a implementação *Gamma-MPI* quanto a *Gamma-GPU* possuem desempenho prejudicado pelos aspectos acima mencionados. Entretanto, *Gamma-GPU* consegue obter um bom desempenho comparado ao *Gamma-MPI*, para grandes entradas, uma vez que a computação necessária a execução de cada reação é realizada de maneira paralela nas *Graphic Processing Units* (GPUs), conforme podemos ver em maiores detalhes no trabalho desenvolvido por Rubens Henrique Pailo de Almeida em [7].

Capítulo 6

Conclusão

6.1 Considerações Finais

Considerando as três implementações de Gamma utilizadas, a saber, *Gamma-Sequencial*, *Gamma-MPI* e *Gamma-GPU*, o presente trabalho forneceu uma implementação do problema de fusão de dados para acompanhamento de contatos, em um ambiente multisensor, baseado nos dois estágios propostos pelo PPDE - Pares de Plots em Dois Estágios. A solução, conforme o PP e o PPDE, baseia-se em grafos para determinar a melhor hipótese de caminho para determinado alvo, identificando assim, quais dos plots recebidos no instante corrente refere-se ao alvo em questão.

Conforme mencionado em [4], o PPDE, em sua concepção original e com sua proposta de processamento de grafos em dois estágios, apresentou desempenho melhor que o método tradicional, o PP, além de fornecer a possibilidade de paralelização da solução, o que não tinha sido implementado até o presente momento, de acordo com a revisão bibliográfica realizada neste trabalho.

Neste contexto, o formalismo Gamma mostrou-se bastante adequado para a representação do problema de fusão de dados aplicado ao acompanhamento de contatos, baseado em uma abordagem onde os plots (dados recebidos pelos sensores que podem refletir um alvo ou algum tipo de ruído) representam vértices de um grafo. Sem dúvida a facilidade em expressar problemas no formalismo Gamma aliado ao paralelismo inerente ao modelo, constituem-se os principais pontos fortes de Gamma.

Assim, implementamos a solução em dois estágios, onde os mesmos foram interligados por um código em linguagem C que, dentre outras coisas, prepara o grafo para um segundo estágio, a partir dos resultados do estágio inicial.

Com os resultados apresentados no Capítulo 5, mais especificamente na Seção 5.3, foram verificados aspectos inerentes à três formas de explorar o paralelismo do problema em questão (Abordagens Sequencial, Paralela e Híbrida). Neste pri-

meiro conjunto de casos de teste ficou evidenciado, para o caso específico de nosso problema, que a melhor alternativa de abordagem de paralelismo em Gamma foi a que explora o paralelismo intra reações, ou seja, as reações foram especificadas para reagir somente com elementos provenientes de um mesmo sensor. Já, quando foram testadas as três implementações de Gamma utilizadas (*Gamma-Sequencial*, *Gamma-MPI* e *Gamma-GPU*) por ocasião da execução de nosso algoritmo, pudemos verificar que a medida em que aumentam-se a quantidade de sensores envolvidos e a quantidade de plots recebidos destes sensores, a implementação *Gamma-GPU* apresenta melhores desempenhos.

É importante mencionar que para os experimentos realizados, no caso específico do problema de fusão de dados para acompanhamento de contatos, dois aspectos justificam os resultados obtidos:

- Para que seja vantajosa a utilização das implementações de Gamma que executam em um ambiente de hardware paralelo, principalmente *Gamma-GPU* que obteve melhor desempenho, o tamanho da entrada deve ser tal que compense o *overhead* de comunicação imposto pela implementação de Gamma. Em nossos casos de teste, a entrada que obteve melhor custo benefício foi a que utilizou 30 plots por sensor em um ambiente composto por 11 sensores, seguida da configuração composta por 9 sensores com 30 plots por sensor e 11 sensores contendo 20 plots cada. Estas três configurações obtiveram *speedups* de 1,80; 1,50 e 1,04 respectivamente; e

- Para *Gamma-MPI* e *Gamma-GPU*, a gerência de execução das reações e distribuição do multiconjunto entre os processadores envolvidos é realizado de maneira centralizada por uma única célula de processamento. Esta sobrecarga computacional em um único núcleo de processamento poderia ser bastante reduzida pela implementação do trabalho sugerido em [42].

Assim, este trabalho forneceu uma implementação de um problema real utilizando-se o formalismo Gamma, através de três implementações existentes. Forneceu a primeira implementação paralela do método PPDE, abordou formas de explorar o paralelismo do algoritmo proposto além de ter realizado a análise de desempenho das implementações de Gamma utilizadas. Com isso, todos os objetivos específicos foram alcançados.

Por fim, acredita-se que este trabalho contribuiu para sugerir melhorias nas implementações de Gamma utilizadas, conforme elencamos na Seção 4.3.6 visando tornar o modelo cada vez mais adequado para a utilização aplicada à problemas reais. Vale mencionar que, procurando facilitar futuros trabalhos nas implementações utilizadas, na Seção 4.1.3 fornecemos informações sobre a sintaxe suportada por estas implementações.

6.2 Lições Aprendidas

Neste período de realização do curso de mestrado inúmeras lições foram aprendidas, muitas delas não relacionadas ao conteúdo técnico. Assim, o contato com o ambiente acadêmico, com os professores e alunos trouxeram diversas lições relacionadas ao amadurecimento profissional, pessoal e acadêmico. Sem dúvida, esta é a maior lição aprendida neste trabalho.

Um ponto que merece destaque neste trabalho é o fato de que é sempre necessário ressaltar que problemas na implementação de algum modelo computacional não refletem necessariamente características do modelo em questão. Em outras palavras, neste trabalho verificamos que três implementações distintas de Gamma apresentaram comportamentos e características distintas entre si. Uma com suas limitações pela utilização de um único processador, outras com limitações inerentes à gerencia de execução em um hardware paralelo. Porém, todas estas limitações referem-se a detalhes de implementação específicos e não fragilidades do formalismo Gamma.

Por fim, uma importante lição aprendida que deseja-se mencionar é conter o ímpeto de implementar sem que seja analisada de maneira adequada todas as possibilidades envolvidas. A implementação do algoritmo deste trabalho passou por diversas versões até que, em determinado momento, decidiu-se abandonar momentaneamente a implementação em prol de um estudo mais aprofundado a respeito da programação a ser realizada. Este fato proporcionou uma facilidade de implementação pela antecipação dos problemas que poderiam existir. Entretanto, a busca por resultados em um modelo computacional promissor onde existem inúmeras possibilidades de trabalho faz com que seja natural a vontade de implementar.

6.3 Trabalhos Futuros

Dentre os trabalhos futuros identificados por ocasião deste estudo, podemos mencionar a realização de testes comparativos aos resultados que obtivemos após a implementação da "*Proposta de um Escalonador para Gamma*" trabalho sugerido por PAILLARD *et al.* [42]. Acreditamos que este trabalho venha alavancar o desempenho das implementações de Gamma que utilizam um ambiente de hardware paralelo para a execução das reações envolvidas, especialmente *Gamma-GPU* que utiliza o mesmo mecanismo para a troca de mensagens da abordagem *Gamma-MPI*, baseado em MPI, porém utiliza os benefícios de processamento das GPU. Atualmente estas duas implementações possuem um gargalo computacional que reside no controle e gestão centralizado da execução de reações e distribuição do multiconjunto. Com a Proposta do Escalonador, tais tarefas seriam realizadas de maneira distribuída pelos processadores envolvidos.

Um segundo trabalho a ser realizado diz respeito a modelagem do problema da fusão de dados para acompanhamento de contatos em uma computação aproximativa. Em outras palavras, a modelagem deverá permitir a obtenção de panoramas da computação em instantes quaisquer adquirindo respostas mais precisas a medida em que o tempo de execução da computação fosse avançando. Ou seja, ao invés da computação ter um fim específico, o interesse nesta computação evolutiva seria utilizar o Gamma como uma máquina para programas sem terminação, no qual estaríamos interessados nos possíveis valores do multiconjunto durante a computação.

Outro trabalho sugerido seria a utilização de bases de dados maiores que 9 sensores com 30 pontos por sensor, visando analisar o comportamento do algoritmo e da implementação *Gamma-GPU* perante entradas extremamente grandes. Além disso, seria interessante analisar o tamanho máximo da janela de amostragem relacionado ao tamanho da entrada, conforme apresentado na Seção 4.2.

Por fim, no âmbito das implementações de Gamma, seria interessante a utilização de uma abordagem que permita a utilização de vários multiconjuntos, além de suporte a valores reais como elementos destes.

Referências Bibliográficas

[1] HENNESSY, J. L., PATTERSON, D. A. *Computer Architecture, Fifth Edition: A Quantitative Approach*. 5th ed. San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2011. ISBN: 012383872X, 9780123838728.

[2] GOLDMAN, A., UEDA, A. H., MATSUBARA, C. M., et al. "Arquitetura de Computadores: educação, ensino e aprendizado". cap. Modelos para Programação em Arquitetura Multi-Core, Sociedade Brasileira de Computação, 2012. ISBN: 978-85-7669-263-8.

[3] BANÂTRE, J.-P., LE MÉTAYER, D. *A New Computational Model and Its Discipline of Programming*. In: Rapport de Recherche 566, INRIA, France, 1986.

[4] BARBOSA, R. P. *Fusão de Alvos Utilizando Grafos em Ambientes de Múltiplos Sensores*. Tese de Doutorado, UFRJ, COPPE, 2012.

[5] LIVERNET, F., CUILLIERE, O. "Tracking based on graph theory applied to pairs of plots". In: *IEEE Radar Conference*, pp. 90–95, 2010.

[6] "The Message Passing Interface (MPI) Standard". Disponível em ¡http://www.mcs.anl.gov/research/projects/mpi/index.htm¿.

[7] DE ALMEIDA, R. H. P. *Uma Derivação do Paradigma de Reescrita de Multiconjuntos Gamma para a Arquitetura GPU*. Dissertação de Mestrado, PESC/COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, 2015.

[8] BANÂTRE, J.-P., LE MÉTAYER, D. "Coordination Programming". Imperial College Press, cap. Gamma and the Chemical Reaction Model: Ten Years After, pp. 3–41, London, UK, UK, 1996. ISBN: 1-86094-023-4. Disponível em: <http://dl.acm.org/citation.cfm?id=270347.270348>.

[9] BANÂTRE, J.-P., LE MÉTAYER, D. "Programming by Multiset Transformation", *Commun. ACM*, v. 36, n. 1, pp. 98–111, jan. 1993. ISSN: 0001-0782.

doi: 10.1145/151233.151242. Disponível em: <http://doi.acm.org/10.1145/151233.151242>.

[10] PAILLARD, G. A. L. *Uma Implementação Paralela e Distribuída de Gamma Estruturada*. Dissertação de Mestrado, PESC/COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, 1999.

[11] BANÂTRE, J.-P., LE MÉTAYER, D. "The Gamma Model and Its Discipline of Programming", *Sci. Comput. Program.*, v. 15, n. 1, pp. 55–77, nov. 1990. ISSN: 0167-6423. doi: 10.1016/0167-6423(90)90044-E. Disponível em: <http://dx.doi.org/10.1016/0167-6423(90)90044-E>.

[12] BANÂTRE, J.-P., FRADET, P., MÉTAYER, D. L. "Gamma and the Chemical Reaction Model: Fifteen Years After". In: *Proceedings of the Workshop on Multiset Processing: Multiset Processing, Mathematical, Computer Science, and Molecular Computing Points of View*, WMP '00, pp. 17–44, London, UK, UK, 2001. Springer-Verlag. ISBN: 3-540-43063-6. Disponível em: <http://dl.acm.org/citation.cfm?id=647269.721851>.

[13] FRADET, P., LE MÉTAYER, D. "Structured Gamma", *Sci. Comput. Program.*, v. 31, n. 2-3, pp. 263–289, jul. 1998. ISSN: 0167-6423. doi: 10.1016/S0167-6423(97)00023-3. Disponível em: <http://dx.doi.org/10.1016/S0167-6423(97)00023-3>.

[14] HANKIN, C., LE MÉTAYER, D., SANDS, D. *A calculus of Gamma programs*. Research Report RR-1758, 1992. Disponível em: <https://hal.inria.fr/inria-00076998>.

[15] SANDS, D. "A Compositional Semantics of Combining Forms for Gamma Programs". In: *International Conference on Formal Methods in Programming and Their Applications*. lncs, 1993.

[16] CIANCARINI, P., GORRIERI, R., ZAVATTARO, G. "An Alternative Semantics for the Calculus of Gamma Programs". In: Andreoli, J., Hankin, C., LeMetayer, D. (Eds.), *Coordination Programming: Mechanisms, Models and Semantics*, Imperial College Press, pp. 232–248, 1996.

[17] MÉTAYER, D. L. "Higher-Order Multiset Programming". In: *in Proc. of the DIMACS workshop on specifications of parallel algorithms, American Mathematical Society, Dimacs series in Discrete Mathematics*. American Mathematical Society, 1994.

[18] BANÂTRE, J. P., FRADET, P., RADENAC, Y. "Higher-Order Chemical Programming Style". In: *Proceedings of the 2004 International Conference on Unconventional Programming Paradigms*, UPP'04, pp. 84–95, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN: 3-540-27884-2, 978-3-540-27884-9. doi: 10.1007/11527800_7. Disponível em: <http://dx.doi.org/10.1007/11527800_7>.

[19] BANÂTRE, J.-P., FRADET, P., RADENAC, Y. "Higher-order Chemical Model of Computation". In: *The Grand Challenge in Non-Classical Computation*, April 2005.

[20] BANÂTRE, J.-P., FRADET, P., RADENAC, Y. "Higher-Order Chemistry". In: *Pre-proceedings of Workshop on Membrane Computing (WMC 2003)*, pp. 53–60, July 2003. Disponível em: <http://www.irisa.fr/paris/Biblio/Papers/Banatre/BanFraRad03WMC.pdf>.

[21] CREVEUIL, C. "Implementation of Gamma on the Connection Machine." In: CREVEUIL [21], pp. 219–230. ISBN: 3-540-55160-3. Disponível em: <http://dblp.uni-trier.de/db/conf/hlppp/hlppp1991.html#Creveuil91>.

[22] HUAN, L. P., NG, K. W., SUN, Y. Q. "Implementing Gamma on the MasPar MP-1", *Journal of Computer Science and Technology*.

[23] BANÂTRE, J.-P., COUTANT, A., METAYER, D. L. "A parallel machine for multiset transformation and its programming style", *Future Generation Computer Systems*, v. 4, n. 2, pp. 133 – 144, 1988. ISSN: 0167-739X. doi: http://dx.doi.org/10.1016/0167-739X(88)90012-X. Disponível em: <http://www.sciencedirect.com/science/article/pii/0167739X8890012X>.

[24] GLADITZ, K., KUCHEN, H. "Parallel Implementation of the Gamma-Operation on Bags". In: *Proceedings of PASCO '94*, pp. 154–163, 1994.

[25] BERRY, G., BOUDOL, G. "The Chemical Abstract Machine". In: *Proceedings of the 17th ACM SIGPLAN-SIGACT annual symposium on principles of programming languages*, pp. 81–94, 1990.

[26] WANG, C., PRIOL, T. *HOCL Programming Guide*. Technical Report hal-00705283, INRIA, France, 2009.

[27] HALL, D. L., LLINAS, J. *Handbook of Multisensor Data Fusion*. New York, USA, CRC Press, 2001. ISBN: 0849323797.

[28] WHITE, J., F.E. *Data Fusion Lexicon*. Joint directors of laboratories, technical panel for c3, data fusion sub-panel, Naval Ocean System Center, San Diego, 1987.

[29] HALL, D. L., MCMULLEN, A. H. *Mathematical Techniques in Multisensor Data Fusion*. 2nd ed. Norwood, MA, USA, Boston: Artech House, 2004.

[30] KOCH, W. "Overview of problems and techniques in target tracking". In: *Target Tracking: Algorithms and Applications (Ref. No. 1999/090, 1999/215), IEEE Colloquium*, pp. 1/1–1/4, 1999.

[31] BAR-SHALOM, Y., WILLETT, P., WILLETT, P., et al. *Tracking and Data Fusion: A Handbook of Algorithms*. USA, YBS Publishing, 2011. ISBN: 9780964831278. Disponível em: <`http://books.google.com.br/books?id=2aOiuAAACAAJ`>.

[32] BLACKMAN, S. S. "Multiple hypothesis tracking for multiple target tracking", *Aerospace and Electronic Systems Magazine, IEEE*, v. 19, n. 1, pp. 5–18, jan. 2004. ISSN: 0885-8985. doi: 10.1109/maes.2004.1263228. Disponível em: <`http://dx.doi.org/10.1109/maes.2004.1263228`>.

[33] BLACKMAN, S. S. *Multiple-target tracking with radar applications*. Artech House radar library. Massachusetts, USA, Norwood, Mass. Artech House, 1986. ISBN: 0-89006-179-3.

[34] TANG, Y., LEE, C. "A geometric feature relation graph formulation for consistent sensor fusion". In: *Systems, Man and Cybernetics, IEEE International Conference*, pp. 188–193, 1990.

[35] GOUVÊA, L. F. Y. *Interceptação de alvos móveis em ambiente com obstáculos poligonais bidimensionais*. Tese de Doutorado, UFRJ, COPPE, 2012.

[36] BARBOSA, R., LIVERNET, F., DE LIMA, B., et al. "Multisensor Data Fusion Using Two-Stage Analysis on Pairs of Plots Graphs". In: *15th International Conference on Informating Fusion, Singapore*, pp. 2073–2078, 2012.

[37] PAILLARD, G., FRANCA, F., FILHO, J. "A distributed implementation of Structured Gamma". In: *Parallel and Distributed Systems, 2001. ICPADS 2001. Proceedings. Eighth International Conference on*, pp. 445–450, 2001. doi: 10.1109/ICPADS.2001.934852.

[38] LEVINE, J., MASON, T., BROWN, D. *Lex & Yacc, 2Nd Edition*. Second ed. California, USA, O'Reilly, 1992. ISBN: 9781565920002.

[39] MURTHY, V., KRISHNAMURTHY, E. “Gamma Programming Paradigm and Heterogeneous Computing”. In: *29th Annual Hawaii International Conference on SystemSciences*, 1996.

[40] MURTHY, V. K., KRISHNAMURTHY, E. V. “Gamma Programming Paradigm and Heterogeneous Computing”. In: *Proceedings of the 29th Hawaii International Conference on System Sciences Volume 1: Software Technology and Architecture*, HICSS ’96, pp. 273–, Washington, DC, USA, 1996. IEEE Computer Society. ISBN: 0-8186-7324-9. Disponível em: <http://dl.acm.org/citation.cfm?id=795698.798419>.

[41] CHRISTIAN CREVEUIL, G. M. *Dérivation systématique d’un algorithme de segmentation d’images: un exemple d’application du formalisme GAMMA*. In: Rapport de Recherche 1049, INRIA, France, 1989.

[42] PAILLARD, G., FRANCA, F., FILHO, J. “Uma Proposta de um Escalonador para Gamma”. In: *II Workshop em Sistemas Computacionais de Alto Desempenho*, pp. 47–54, Pirenópolis, GO, 2001.

Apêndice A

Apêndice

Exemplo de código do Primeiro Estágio para uma aplicação com 3 sensores, 10 plots por sensor.

```
/*Reações*/

R1;R2;R3;R4;R5;R6;R7;R8;R9;R10;R11;R12;R13;R14;R15;R16
{


/*Sensor Número 1*/


[1,1,0,0,0,0],[1,2,3,4,10,0],[1,3,5,5,20,0],[1,4,7,6,30,0],
[1,5,6,6,30,0],[1,6,5,6,30,0],[1,7,6,7,40,0],[1,8,5,7,40,0],
[1,9,9,8,50,0],[1,10,3,8,50,0],

[1,1,1,0,0],[1,1,2],[1,1,3],[1,1,4],[1,1,5],[1,1,6],[1,1,7],[1,1,8],[1,1,9],[1,1,10],
[1,2,1],[1,2,2],[1,2,3],[1,2,4],[1,2,5],[1,2,6],[1,2,7],[1,2,8],[1,2,9],[1,2,10],
[1,3,1],[1,3,2],[1,3,3],[1,3,4],[1,3,5],[1,3,6],[1,3,7],[1,3,8],[1,3,9],[1,3,10],
[1,4,1],[1,4,2],[1,4,3],[1,4,4],[1,4,5],[1,4,6],[1,4,7],[1,4,8],[1,4,9],[1,4,10],
[1,5,1],[1,5,2],[1,5,3],[1,5,4],[1,5,5],[1,5,6],[1,5,7],[1,5,8],[1,5,9],[1,5,10],
[1,6,1],[1,6,2],[1,6,3],[1,6,4],[1,6,5],[1,6,6],[1,6,7],[1,6,8],[1,6,9],[1,6,10],
[1,7,1],[1,7,2],[1,7,3],[1,7,4],[1,7,5],[1,7,6],[1,7,7],[1,7,8],[1,7,9],[1,7,10],
[1,8,1],[1,8,2],[1,8,3],[1,8,4],[1,8,5],[1,8,6],[1,8,7],[1,8,8],[1,8,9],[1,8,10],
[1,9,1],[1,9,2],[1,9,3],[1,9,4],[1,9,5],[1,9,6],[1,9,7],[1,9,8],[1,9,9],[1,9,10],
[1,10,1],[1,10,2],[1,10,3],[1,10,4],[1,10,5],[1,10,6],[1,10,7],[1,10,8],[1,10,9],
[1,10,10],
```

```
/* Sensor Número 2 */


[2,1,0,0,0,0],[2,2,4,12,10,0],[2,3,5,10,10,0],[2,4,7,16,10,0],
[2,5,7,12,10,0],[2,6,8,8,20,0],[2,7,10,7,30,0],[2,8,11,5,30,0],
[2,9,9,10,40,0],[2,10,10,17,50,0],

[2,1,1,0,0],[2,1,2],[2,1,3],[2,1,4],[2,1,5],[2,1,6],[2,1,7],[2,1,8],[2,1,9],[2,1,10],
[2,2,1],[2,2,2],[2,2,3],[2,2,4],[2,2,5],[2,2,6],[2,2,7],[2,2,8],[2,2,9],[2,2,10],
[2,3,1],[2,3,2],[2,3,3],[2,3,4],[2,3,5],[2,3,6],[2,3,7],[2,3,8],[2,3,9],[2,3,10],
[2,4,1],[2,4,2],[2,4,3],[2,4,4],[2,4,5],[2,4,6],[2,4,7],[2,4,8],[2,4,9],[2,4,10],
[2,5,1],[2,5,2],[2,5,3],[2,5,4],[2,5,5],[2,5,6],[2,5,7],[2,5,8],[2,5,9],[2,5,10],
[2,6,1],[2,6,2],[2,6,3],[2,6,4],[2,6,5],[2,6,6],[2,6,7],[2,6,8],[2,6,9],[2,6,10],
[2,7,1],[2,7,2],[2,7,3],[2,7,4],[2,7,5],[2,7,6],[2,7,7],[2,7,8],[2,7,9],[2,7,10],
[2,8,1],[2,8,2],[2,8,3],[2,8,4],[2,8,5],[2,8,6],[2,8,7],[2,8,8],[2,8,9],[2,8,10],
[2,9,1],[2,9,2],[2,9,3],[2,9,4],[2,9,5],[2,9,6],[2,9,7],[2,9,8],[2,9,9],[2,9,10],
[2,10,1],[2,10,2],[2,10,3],[2,10,4],[2,10,5],[2,10,6],[2,10,7],[2,10,8],[2,10,9],
[2,10,10],



/* Sensor Número 3 */


[3,1,0,0,0,0],[3,2,3,4,10,0],[3,3,5,5,20,0],[3,4,8,8,20,0],
[3,5,9,12,30,0],[3,6,9,13,30,0],[3,7,9,16,40,0],[3,8,8,14,40,0],
[3,9,9,19,40,0],[3,10,10,15,50,0],

[3,1,1,0,0],[3,1,2],[3,1,3],[3,1,4],[3,1,5],[3,1,6],[3,1,7],[3,1,8],[3,1,9],[3,1,10],
[3,2,1],[3,2,2],[3,2,3],[3,2,4],[3,2,5],[3,2,6],[3,2,7],[3,2,8],[3,2,9],[3,2,10],
[3,3,1],[3,3,2],[3,3,3],[3,3,4],[3,3,5],[3,3,6],[3,3,7],[3,3,8],[3,3,9],[3,3,10],
[3,4,1],[3,4,2],[3,4,3],[3,4,4],[3,4,5],[3,4,6],[3,4,7],[3,4,8],[3,4,9],[3,4,10],
[3,5,1],[3,5,2],[3,5,3],[3,5,4],[3,5,5],[3,5,6],[3,5,7],[3,5,8],[3,5,9],[3,5,10],
[3,6,1],[3,6,2],[3,6,3],[3,6,4],[3,6,5],[3,6,6],[3,6,7],[3,6,8],[3,6,9],[3,6,10],
[3,7,1],[3,7,2],[3,7,3],[3,7,4],[3,7,5],[3,7,6],[3,7,7],[3,7,8],[3,7,9],[3,7,10],
[3,8,1],[3,8,2],[3,8,3],[3,8,4],[3,8,5],[3,8,6],[3,8,7],[3,8,8],[3,8,9],[3,8,10],
[3,9,1],[3,9,2],[3,9,3],[3,9,4],[3,9,5],[3,9,6],[3,9,7],[3,9,8],[3,9,9],[3,9,10],
[3,10,1],[3,10,2],[3,10,3],[3,10,4],[3,10,5],[3,10,6],[3,10,7],[3,10,8],[3,10,9],
[3,10,10]
}
```

```
where


R1 = replace [ids,ida,idb],[ids1,ida1,idb1]
by [ids1,ida1,idb1]
if ids == ids1 and ida == idb and ida != 1
/* R1 - Retira as arestas no formato [ids,x,x]. */
/* Ou seja, retira arestas de um nó para o mesmo nó. */
/* Com exceção da aresta [x,1,1,0,0] */


R2 = replace [ids,idv,x,y,tempo,marca], [ids1,idv1,x1,y1,tempo1,marca1],
[ids2,ida,idb]
by [ids,idv,x,y,tempo,marca], [ids1,idv1,x1,y1,tempo1,marca1]
if ids == ids1 and ids1 == ids2 and ida == idv and idb == idv1
and tempo1 <= tempo
/* R2 - Retira arestas que fazem referência a caminhos no passado.*/


R3 = replace [ids,idv,x,y,tempo,marca], [ids1,idv1,x1,y1,tempo1,marca1],
[ids2,ida,idb]
by [ids,idv,x,y,tempo,marca], [ids1,idv1,x1,y1,tempo1,marca1]
if ids==ids1 and ids1==ids2 and ida==idv and idb==idv1
and tempo1 > tempo and tempo1-tempo > 10
/* R3 - Retira as arestas que não satisfizerem a condição */
/* de consistência cinemática  */


R4 = replace [ids,idv,x,y,tempo,marca], [ids1,idv1,x1,y1,tempo1,marca1],
[ids2,ida,idb]
by [ids,idv,x,y,tempo,marca],[ids1,idv1,x1,y1,tempo1,marca1],
[ids2,ida,idb,tempo1-idv1,32000]
if ids==ids1 and ids1==ids2 and ida==idv and idb==idv1
and tempo1 > tempo and tempo1-tempo <= 10
/* R4 - Transforma as arestas válidas (as que podem ser */
/* alcançadas dada a velocidade máxima inicial), */
/* inserindo o parâmetro de velocidade */
```

```
R5 = replace [ids,r,y,c,m],[ids1,r1,y1,c1,m1]
by [ids,r,y,c,c],[ids1,r1,y1,c1,m1]
if ids == ids1 and r == 1 and m == 32000 and r != y
and (r1 and y1) == 1 and (c1 and m1) == 0
/* R5 - Primeiro passo do algoritmo que calcula o melhor caminho. */
/* Atribui o custo as arestas adjacentes ao no origem */


R6 = replace [ids,x,y,c1,s1],[ids1,y1,z,c2,s2]
by [ids,x,y,c1,s1],[ids1,y1,z,c2,s1+c2]
if ids == ids1 and y == y1 and s2 > (s1+c2)
/* R6 - Segundo passo do calculo de melhor caminho. */
/* Atribui o custo das demais arestas */


R7 = replace [ids,x,y,c1,s1],[ids1,z,y1,c2,s2]
by [ids,x,y,c1,s1]
if ids == ids1 and y == y1 and s1 < s2
/* R7 - Finalização do calculo do custo do melhor caminho. */
/* Retira arestas que chegam a um mesmo no, mantendo somente */
/* a de menor custo */


R8 = replace [ids,r,y,c,m], [ids1,id,x1,y1,tempo,marca]
by [ids,r,y,m],[ids1,id,x1,y1,tempo,marca],[ids,r],[ids,y]
if ids == ids1
/* R8 - Insere todos os identificadores (ids,identificador) */
/* dos elementos nós que possuem arestas chegando ou saindo. */
/* E reduz um elemento da tupla do tipo aresta */


R9 = replace [ids,x],[ids1,y]
by [ids,x]
if ids == ids1 and x > y
/* R9 - Seleciona o nó de maior id, alcançável pelo grafo */


R10 =replace [ids,x], [ids1,r,y,m]
by [ids,r], [ids1,r,y,m,32000]
if ids == ids1 and x == y
/* R10 - Marca todas as arestas do melhor caminho */
```

```
R11 = replace [ids,r,y,m],[ids1,r1,y1,m1,32000]
by [ids1,r1,y1,m1,32000]
if ids == ids1
/* R11 - Exclui as arestas que não fazem parte do melhor caminho */


R12 = replace [ids,r,y,m,s],[ids1,idv,x,y1,tempo,marca]
by [ids,r,y,m,s],[ids1,idv,x,y1,tempo,1]
if ids == ids1 and marca != 1 and (r == idv or y == idv)
/* R12 - Marca todos os nós que fazem parte do melhor caminho */
/* Nós que são entrada ou saída de alguma aresta */


R13 = replace [ids,idv,x,y,tempo,marca], [ids1,idv1,x1,y1,tempo1,marca1]
by [ids,idv,x,y,tempo,marca]
if ids == ids1 and marca1 == 0 and marca == 1
/* R13 - Exclui todos os nós que não fazem parte do melhor */
/* caminho escolhido */


R14 = replace [ids,idv,x,y,tempo,marca], [ids1,r1,y1,m1,32000]
by [ids,idv,x,y,tempo,marca]
if ids == ids1
/* R14 - Exclui todas as arestas, para o início do segundo estágio */


R15 = replace [ids,idv,x,y,tempo,marca], [ids1,aux]
by [ids,idv,x,y,tempo,0], [ids1,aux]
if ids == ids1 and marca == 1
/* R15 - Desmarca os nós selecionados visando o próximo estágio */


R16 = replace [ids,idv,x,y,tempo,marca], [ids1,aux]
by [ids,idv,x,y,tempo,marca]
if ids == ids1
/* R16 - Exclui o elemento unitário - auxiliar */
/* Assim, tem-se um multiconjunto pronto para o segundo estágio */
/* Ou seja, um multiconjunto com os nós selecionados pelo primeiro estágio */
```