Análise das Causas da Perda de Desempenho da MFDM e Possível Solução

O Impacto do Escalonamento de Instruções

Paulo A. R. Lorenzo

Análise das Causas da Perda de Desempenho da MFDM
e
Possível Solução
O Impacto do Escalonamento de Instruções

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. *Paulo A. R. Lorenzo* e *aprovada pela Comissão Julgadora.*

Campinas, 28 de abril de 1995.

Prof. Dr. Arthur J. Catto
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, *como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.*

FICHA CATALOGRAFICA ELABORADA PELA
BIBLIOTECA DO IMECC DA UNICAMP

Lorenzo, Paulo Adelino Rosario

L887a Analise das causas da perda de desempenho da MFDM e possivel solucao : o impacto do escalonamento de instrucoes / Paulo Adelino Rosario Lorenzo. -- Campinas, SP : s.n.], 1995.

Orientador : Arthur Joao Catto.

Dissertacao (mestrado) - Universidade Estadual de Campinas, Instituto de Matematica, Estatistica e Ciencia da Computacao.

1. Fluxo de dados (Computacao). 2. Agenda de execucao (Administracao). I. Catto, Arthur Joao. II. Universidade Estadual de Campinas. Instituto de Matematica, Estatistica e Ciencia da Computacao. III. Titulo.

Análise das Causas da Perda de Desempenho da MFDM
e
Possível Solução

O Impacto do Escalonamento de Instruções[1]

Paulo A. R. Lorenzo[2]

Departamento de Ciência da Computação
IMECC – UNICAMP

Banca Examinadora:

- Arthur J. Catto (Orientador)[3]
- Carlos Antônio Ruggiero[4]
- Célio Cardoso Guimarães[3]
- Maria Beatriz Felgar de Toledo (Suplente)[3]

---

[1] Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

[2] O autor é Bacharel em Processamento de Dados pela Universidade Federal da Bahia (UFBA).

[3] Professor(a) do Departamento de Ciência da Computação - IMECC - UNICAMP.

[4] Professor da USP de São Carlos.

Tese defendida e aprovada em, 28 de abril de 1995

Pela Banca Examinadora composta pelos Profs. Drs.

Prof(a). Dr(a). CARLOS ANTONIO RUGGIERO

Prof(a). Dr(a). CÉLIO CARDOSO GUIMARÃES

Prof(a). Dr(a). ARTHUR JOÃO CATTO

Aos meus Pais,
José e Olga Fernanda.

Agradecimentos

"Amigo é aquela pessoa que sabe tudo a seu respeito,
mas gosta de você assim mesmo."

Aos meus Pais que sempre me ensinaram o valor do conhecimento. E aos meus Familiares por estarem sempre ao meu lado desde de antes do meu nascimento.

Ao meu Sensei, Prof. Dr. Munehiro Goto, pelo constante apoio, presença e orientação.

Aos Amigos Conspiradores:

Carlinhos, Carlos Cordeiro de Melo Neto, Amigo e pianista de sempre.
Carrilho, José Aparecido Carrilho, por ser meu Amigo de todos os momentos e procurador.
Mivs, Maria Inês Vale da Silva, Amiga e companheira de diárias conversas via correio eletrônico.
Nilza, Nilza Mizosoe, Amiga de sempre e de longas conversas via telefone.

Aos meus amigos do oriente: Katsuya Tsunemi, Chen Song, Tsuyoshi Kimura, Santosa Slamet e Shinya Kaneko. Amizade não tem fronteiras.

A Amiga Vilmara, Vilmara Rosa Piccoli, pelo apoio, sempre.

Ao Amigo Walter, Walter Rocha Palma, pelas longas conversas.

A Fred, Frederico Sidney Cox Júnior, meu Orientador informal de curso.

Ao Sempai Carlos, Carlos Eduardo Jeronymo, por estar sempre por perto no Japão.

Ao pessoal do DCC e CORI da UNICAMP, São Paulo, Brasil.

Ao pessoal do DCC e da secretaria para assuntos internacionais da Universidade de Gifu, Gifu, Japão.

Ao apoio financeiro do CNPq, Brasil, e da associação Nakashima Zaidan, Japão.

E, ao meu orientador, Prof. Dr. Arthur João Catto, por ter-me aceito como seu orientando.

"Jamais desista daquilo que você quer realmente fazer. A pessoa que tem grandes sonhos é mais forte do que aquela que possui todos os fatos."

"Não deixe ninguém convencê-lo a desistir daquilo que você sabe que é uma grande idéia."

"Exerça liderança: lembre-se de que o primeiro cão de neve que puxa o trenó é o único que desfruta de uma vista decente."

"Qualquer pessoa que você conheça sabe alguma coisa que você não sabe, mas precisaria saber. Aprenda com elas."

"Ninguém consegue ser sábio com o estômago vazio. Alimente-se."

"Nunca se aprendeu nada que valesse a pena em 10 lições fáceis."

Todas as citações existentes nesta página e no decorrer deste texto são colaboração da amiga e conspiradora, a artista plástica *Nilza Mizosoe.*

Resumo

Na busca de propostas mais eficientes para se alcançar altos níveis de paralelismo, o Modelo de Fluxo de Dados (MFD) emergiu como um caminho novo e promissor a ser seguido. O MFD advoga uma representação clara e uma manipulação fácil do paralelismo dos programas. Dentre as pesquisas que abordam computação por fluxo de dados, a Máquina de Fluxo de Dados de Manchester (MFDM) ocupa uma posição importante, uma vez que ela foi uma das primeiras máquinas de fluxo de dados a serem projetadas e construídas.

O Projeto de Fluxo de Dados de Manchester aventurou-se por um novo caminho e, durante mais de uma década, atingiu muitos dos seus objetivos. Dentre esses, destacam-se análises pioneiras de um sistema baseado no MFD. Nessas análises, duas medidas, Pby e PM, foram largamente utilizadas como indicadores consistentes do comportamento da Unidade de Emparelhamento (UE) e do desempenho das execuções no sistema, respectivamente, apesar de elas serem medidas "brutas".

Embora não se negue aqui a importância dessas duas medidas, mostra-se que falhas na sua aferição podem ter causado deturpado análises anteriores. Nesta dissertação, demonstra-se que as medidas Pby e PM não são indicadores tão consistentes como se admitia. Infelizmente, não se propõe qualquer medida alternativa. Entretanto, pode-se concluir que a análise do comportamento e do desempenho da MFDM deveria ser conduzida com parâmetros mais dinâmicos.

Nos primeiros estágios do projeto da MFDM, observou-se uma perda de desempenho sensível. Desde então, no intuito de corrigir problemas iniciais, vários estudos foram realizados. Esta dissertação aborda especialmente o desempenho do escalonamento de instruções na MFDM.

A técnica de escalonamento da MFDM usa a política "primeiro a chegar, primeiro a sair" (FIFO), que ordena os dados com base na ordem de chegada dos pacotes. Demonstra-se aqui que a técnica FIFO satura a Unidade de Emparelhamento (UE) produzindo seqüências indesejáveis de pacotes, as quais reduzem o *throughput* da UE e, conseqüentemente, o desempenho do sistema. A UE é apontada nas publicações afins como o "gargalo" do sistema. Aqui se apresenta uma análise dos efeitos da substituição da técnica de escalonamento FIFO por técnicas mais elaboradas. Um simulador da MFDM, gMDMS, foi implementado durante este estudo com o objetivo de comparar o desempenho da técnica FIFO com outras técnicas de escalonamento conhecidas, como HLFNET e CP/MISF.

Os resultados obtidos utilizando-se gMDMS permitem concluir que a adoção da técnica FIFO é uma causa indireta da perda real de desempenho apresentada

pela MFDM. Por isso, a especialização do controle da ordem do fluxo dos dados pelo anel provoca uma grande melhora da utilização das unidades da máquina e o conseqüente aumento do desempenho do sistema.

Abstract

In the research for more efficient approaches to reach very high levels of parallelism, DFM (Dataflow Model) emerged as a new and thriving way to follow. DFM advocates a very clear representation and easy manipulation of program parallelism. Among the researches on dataflow computation, the MDFM (Manchester Dataflow Machine) takes an important place, because it is one of the first dataflow machines designed and built.

Manchester Dataflow Project did venture a new pathway, and did achieve many of its expected goals over more than a decade. One of the greatest contributions of this machine project is the set of pioneering analyses of a dataflow system. In these analyses two measures, Pby and AP, were widely used as consistent indicators of the Matching Unit behavior and the system execution performance, respectively, despite their crude nature.

We do not deny the importance of those two measures, but we show here that a deficiency in checking up on the effectiveness of them may have caused some misunderstandings in the previous analyses. In this thesis we show that those two measures, Pby and AP, are not so consistent indicators as they were stated to be. Unfortunately, we could not propose any sound alternative measures. But we can conclude that the analysis of the behavior or performance of MDFM should take place more dynamically.

At the first stages of project, it was pointed out that the MDFM showed a lack of performance. Since then, to overcome the initial problems, several studies have been conducted. This thesis specially subjects the instruction scheduling performed in the MDFM.

The MDFM scheduling technique uses the first-in-first-out (FIFO) policy. The FIFO technique orders the entries based on the packet arriving order. We believe this technique overloads the Matching Unit (MU) by producing undesirable token packet sequences, which decrease the MU throughput and, consequently, the system performance. The MU is pointed out as the bottleneck of the system in the related publications. An analysis of the effects of changing FIFO scheduling technique to a more elaborate one is here performed. For the purpose of comparison, some well-known scheduling techniques, such as HLFNET and CP/MISF, are simulated in the MDFM simulator gMDMS.

We conclude that FIFO technique is an indirect cause of the actual loss of performance seen in the MDFM. Therefore, specializing the controlling of the data flow order in the system ring provides a great improvement in the unit utilization, and consequently an improvement of the system performance.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

"Explore o desconhecido."

A crescente busca de potência computacional tem dirigido esforços à computação paralela. Neste percurso, tanto a adaptação do modelo von Neumann convencional, como o desenvolvimento de modelos de natureza paralela, têm sido considerados. Dentre os modelos paralelos, destaca-se o Modelo de Fluxo de Dados[1] (MFD), baseado no controle da execução através do fluxo dos dados entre as instruções. Seu conceito simples elimina muitos dos problemas encontrados na paralelização do modelo von Neumann.

Devido à grande distância entre o MFD e os já existentes, um grande trabalho de base teve que ser realizado. Métricas de avaliação tiveram que ser adotadas e validadas. Diferentes estruturas de execução e técnicas de controle tiveram que ser desenvolvidas. Dentre os projetos derivados do conceito de fluxo de dados, o da Máquina de Fluxo de Dados de Manchester[2] (MFDM), desenvolvida, simulada e prototipada na Universidade de Manchester, Inglaterra, foi um dos pioneiros e deu grandes contribuições à área.

Entretanto, a MFDM apresenta algumas decisões de projeto que comprometeram seu desempenho. Seus resultados não se mostraram suficientes para superar o rápido aperfeiçoamento das arquiteturas paralelas baseadas no modelo von Neumann. Algumas reavaliações são necessárias para o seu aperfeiçoamento e, dentre elas, pode-se destacar a da técnica de escalonamento de tarefas.

[1] *Dataflow Model.*
[2] *Manchester Dataflow Machine.*

O escalonamento eficiente de processos é um dos problemas críticos em multiprocessadores. Ele engloba a partição do programa em módulos e a alocação destes nas Unidades de Processamento[3] (UP). Problemas como a determinação do tamanho ideal dos módulos e a análise dos custos de comunicação ainda não tiveram soluções satisfatórias. Técnicas genéricas não produziram grandes resultados, devido à correlação entre as questões pontos e as arquiteturas alvo. Parâmetros como o número de UPs e características da rede de intercomunicação têm que ser considerados em qualquer proposta de escalonamento.

A MFDM apresenta uma constante perda de desempenho durante a maioria de suas execuções. Também é observada uma grande utilização do dispositivo de armazenamento de dados da Unidade de Emparelhamento, muito superior ao inicialmente esperada. Por estas razões, tem-se considerado esta unidade como o problema comum das execuções e como a principal causa da perda de desempenho. Esta dissertação é um passo no sentido da reavaliação e identificação das causas desta perda e na análise do impacto no sistema da especialização do escalonamento de tarefas.

1.1 Problemas em Sistemas Paralelos

Existem três problemas fundamentais a serem solucionados na execução de programas paralelos em um multiprocessador: identificar o paralelismo do programa, determinar a sua partição em módulos — ou grãos — e escaloná-los entre as UPs [Sar89]. O primeiro problema é de responsabilidade da linguagem de programação [Bac78, Ack82, Sar89]. Os outros estão relacionados com parâmetros da arquitetura em questão, como o número de UPs, a sobrecarga de sincronização e os custos de comunicação [ERL90, KL88].

A partição do programa é necessária para garantir que a granularidade do programa seja suficiente para manter um mínimo de paralelismo num multiprocessador [Sar89]. Se o grão for muito grande, o paralelismo é limitado; se o grão for muito pequeno, os atrasos de comunicação reduzem o desempenho [KL88].

O escalonamento dos módulos é necessário para manter uma boa utilização das UPs e otimizar a comunicação entre elas [Sar89, KL88]. Contudo, o problema de escalonar n tarefas para p UPs é NP-completo [Ull75, Cof76, GJ79, LER92]. Com o objetivo de se alcançar o melhor desempenho possível, vários trabalhos têm tratado deste tema [ERL90, KL88, Sar89, YSK91, VBJ90, LHCA88, Tow86]. Contudo, qual o melhor tamanho de cada grão, como dividir o programa em

[3] O termo "unidade de processamento" será utilizado como sinônimo de "processador" e de "elemento de processamento".

módulos concorrentes e como obter o mais eficiente escalonamento, para se conseguir o menor tempo de execução possível, são questões que ainda demandam pesquisas.

Os problemas de partição e de escalonamento não são tratados isoladamente, tendo em geral uma análise conjunta. As técnicas de escalonamento normalmente englobam a partição do programa.

Uma outra propriedade que deve ser analisada em multiprocessadores é a relação entre o aumento dos recursos disponíveis e o desempenho da arquitetura [AI87]. Ao contrário do esperado, por exemplo, um aumento do número de UPs pode causar um aumento no tempo para se completar uma tarefa [Gra72].

1.2 Linguagens de Programação Paralela

Simultaneidade é considerada a chave para computação de alto desempenho. Entretanto, ela pode ser reduzida devido a três tipos de dependências: de dados, de controle e de recursos [GPK82, Das89].

Linguagens de programação convencionais são basicamente versões mais abstratas e complexas do computador von Neumann [Bac78]. Muitas dessas linguagens ou dialetos não são naturais, por não refletirem a maneira como os programadores normalmente pensariam para resolver um problema [Ack82]. Devido à origem seqüencial das linguagens convencionais, vários cuidados e técnicas têm que ser adotados para sua utilização em arquiteturas paralelas [Das89].

As propriedades que os computadores de fluxo de dados requerem das linguagens são benéficas para o desempenho e para elas mesmas. São, por outro lado, muito semelhantes a algumas propriedades (p. ex., controle disciplinado de estruturas) conhecidas para facilitar a compreensão e a manutenção de sistemas convencionais [Ack82]. A propriedade de *single assignment*, por exemplo, elimina as dependências de dados artificiais [Cha71, Ack82, Das89].

Linguagens de fluxo de dados formam uma subclasse de linguagens baseadas principalmente em aplicações funcionais (i.e., linguagens aplicativas) [DK82]. Nestas linguagens, entre outros aspectos positivos, não existem maneiras para expressar construções com efeitos colaterais [DK82, Das89, Ack82, Cha71].

1.3 Modelo von Neumann

Uma característica importante do modelo von Neumann é a existência de um contador de instruções, que contém o endereço da próxima instrução a ser executada [AA82]. Por isto, o fluxo de controle é implicitamente seqüencial, embora seja

tolerável o uso explícito de operadores de controle que proporcionam paralelismo [TBH82].

Numa forma simplificada, um computador von Neumann é composto de três partes: uma unidade de execução (CPU), uma memória e um canal de comunicação, que transmite informações entre a CPU e a memória. As informações transmitidas compreendem dados, instruções e endereços [Bac78].

A memória contém o programa e os dados, sendo que as instruções do programa alteram freqüentemente o conteúdo da memória durante a execução [AA82]. Os dados são passados indiretamente entre as instruções via referências à memória [TBH82]. Assim, uma grande parte do tráfego no canal de comunicação é devida à transmissão de informações auxiliares, inerentes ao modelo [Bac78].

Distribuir a carga gerada pela execução de um programa entre UPs von Neumann, paralelizando seus módulos, acarreta uma sobrecarga de sincronização, devido ao controle de consistência dos dados compartilhados. Para amenizar este custo, a escolha dos módulos deve ser otimizada. Embora conseguida satisfatoriamente no caso de acesso regular a estruturas de dados seqüenciais, nos casos mais gerais a solução deste problema permanece uma tarefa difícil [AA82, Das89].

As pesquisas têm privilegiado duas vertentes na solução dos problemas de tamanho do grão e do escalonamento destes [KL88, Sar89]: listas de escalonamento (p. ex., *load balancing*) [Gra72] e *large-grain dataflow* [Bab84]. Contudo, a análise dos custos de comunicação entre UPs ou não é realizada (p. ex., listas de escalonamento), ou acarreta o não aproveitamento de todo o paralelismo existente no programa (p. ex., *large-grain dataflow*). Algumas vezes, módulos interdependentes, com altos custos de comunicação, escalonados para processadores diferentes, poderiam ser mais eficientemente escalonados para um mesmo processador [KL88].

1.4 Modelo de Fluxo de Dados

No modelo de fluxo de dados não existe a noção de um único ponto ou *locus* de controle, o que acarreta um assincronismo natural na execução de instruções [Den85]. Os dados fluem de instrução para instrução e a disponibilidade dos operandos de uma instrução é que determina a sua execução. Os fluxos de dados e de controle são idênticos [TBH82] e o sincronismo entre as instruções é garantido implícita e minimamente pelo percurso dos dados [Das89], ao contrário do que ocorre no modelo de von Neumann, onde os dados permanecem estáticos num endereço de memória. O modelo de fluxo de dados é um exemplo de computação

orientada pelos dados[4] [TBH82].

Programas no modelo de fluxo de dados são usualmente descritos por grafos orientados, chamados Grafos de Fluxo de Dados[5] (GFD) [TBH82, DK82]. Num GFD, cada vértice representa uma instrução e cada aresta a dependência de dados entre duas instruções. A Figura 1.1 apresenta um trecho de programa e sua representação como um GFD. Esta representação destaca um paralelismo de granularidade fina, ao nível de instrução [Das89].

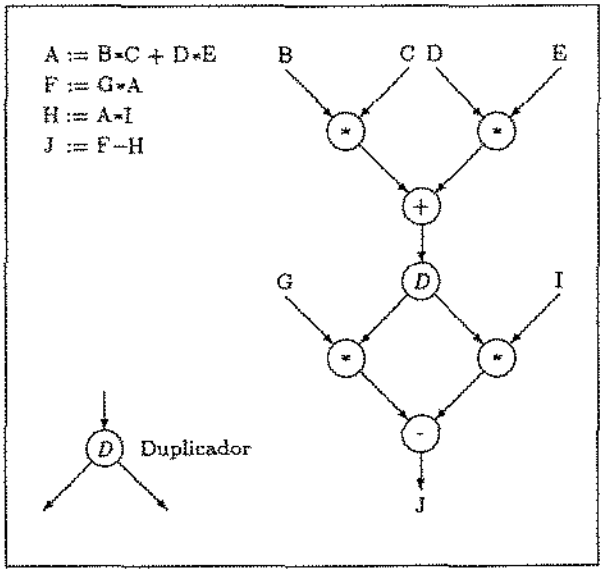

Figura 1.1: Um trecho de programa e seu Grafo de Fluxo de Dados

Considerando que as arestas são como tubos por onde os dados fluem de uma tarefa para outra, é fácil observar que não é necessário o armazenamento dos dados. Após a sua produção, o dado deve fluir da tarefa produtora direto para a tarefa consumidora (ver Figura 1.2). Este procedimento evita os problemas gerados pela necessidade de acesso à memória para manipulação das variáveis: atualização e consulta.

No MFD estático[6], uma instrução só pode ser executada quando todos os seus operandos estiverem disponíveis e suas arestas de saída vazios [Den85]. Assim,

[4] *Data-driven computation.*
[5] *Data Flow Graph.*
[6] *Static dataflow model.*

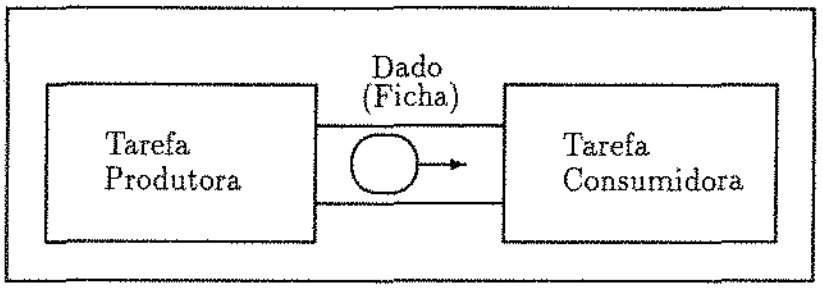

Figura 1.2: Dado fluindo da tarefa produtora para a consumidora.

só pode haver um dado em cada aresta por vez. Já no modelo dinâmico[7], não existe essa restrição sobre as arestas de saída. Ou seja, existindo os operandos, a instrução pode ser executada [Das89], podendo, então, co-existir mais de um dado por aresta. Em ambos os modelos, a execução de uma instrução resulta no consumo dos operandos e na produção de novos dados nas arestas de saída[8].

Ambos modelos, estático e dinâmico, adotam o princípio de *eager evaluation*: as instruções ficam prontas para execução desde que todos os seus argumentos estejam disponíveis e todas as instruções prontas para execução são processadas [VBJ90, TBH82]. Este processo "faminto" torna as execuções assíncronas e as antecipa ao máximo possível. Entretanto, não se leva em conta que os recursos de processamento são limitados e que instruções cujos resultados não serão imediatamente aproveitados possam ocupar o lugar de instruções do caminho crítico [VBJ90]. Isto pode afetar severamente o desempenho da máquina.

Partição e escalonamento são pontos que devem ser analisados na busca de uma solução para esse problema.

1.4.1 Partição

Os GFDs ressaltam, também, a intensa comunicação entre instruções imposta pelo MFD, o que torna a rede de comunicação um gargalo potencial [YSY90]. Por mais paralelismo que exista no ambiente, a intensidade do fluxo de dados entre instruções, que se reflete numa freqüente comunicação entre as UPs, é um fator limitante do desempenho do sistema.

[7] *Dynamic dataflow model.*

[8] Para detalhes sobre as implicações dos modelos estático e dinâmico, recomendam-se as referências [Das89, Den85, WG82].

No caso de trechos seqüenciais, a paralelização só aumenta o tempo de execução, devido aos custos desnecessários de comunicação. O empacotamento desses trechos num só módulo pode reduzir o tempo de execução, aumentando o desempenho do sistema [Vis, YSK91, KL92b].

A partição do programa deve ser realizada com base nas dependências explícitas do GFD e na análise dos custos de comunicação entre instruções (ver 4.3), parâmetro este vinculado à arquitetura em questão. Nesta abordagem, a variabilidade do tamanho dos módulos evita a paralelização prejudicial [VC92, KL92b].

1.4.2 Escalonamento

Com base na partição do programa e nos parâmetros da máquina, é realizado o escalonamento dos módulos. Ele pode ocorrer em tempo de compilação (estático), em tempo de execução (dinâmico) ou em ambos (ver 4.2.1) [CK88].

No MFD, esperava-se que a simples dinâmica de execução dos grafos levasse as implementações a um escalonamento adequado. Contudo, devido aos diferentes tempos de execução das instruções, às concessões ao modelo em cada implementação e ao limitado número de UPs, entre outros fatores, o escalonamento implícito não obtém os resultados desejados.

O elemento que seleciona a UP para a próxima instrução na maioria das implementações do modelo de fluxo de dados tem um algoritmo simples, como o de um banco com fila única de atendimento para vários caixas. Contudo, a aplicação de técnicas mais elaboradas de escalonamento pode representar um grande ganho para o sistema [YSK91, VBJ90, KL88, ERL90].

Por exemplo, considere a Figura 1.3(a), que mostra um GFD representando um trecho de programa. Supondo que o tempo de execução de todos os vértices seja o mesmo e que não haja atrasos de comunicação, a execução desse trecho, numa arquitetura com duas UPs, segundo o MFD puro, poderia ser representada pelo *Gantt chart* da Figura 1.3(b). Pode-se observar facilmente que um escalonamento mais rigoroso pode produzir um melhor resultado, como o da Figura 1.3(c).

1.5 Máquina de Fluxo de Dados de Manchester

A MFDM é estruturada como um conjunto de unidades interligadas em anel. Seguindo o MFD, a comunicação entre as instruções é realizada via o fluxo dos dados. A existência dos dados é que habilita as instruções para execução. As instruções habilitadas disputam as UPs de acordo com a ordem em que elas chegam na unidade de execução.

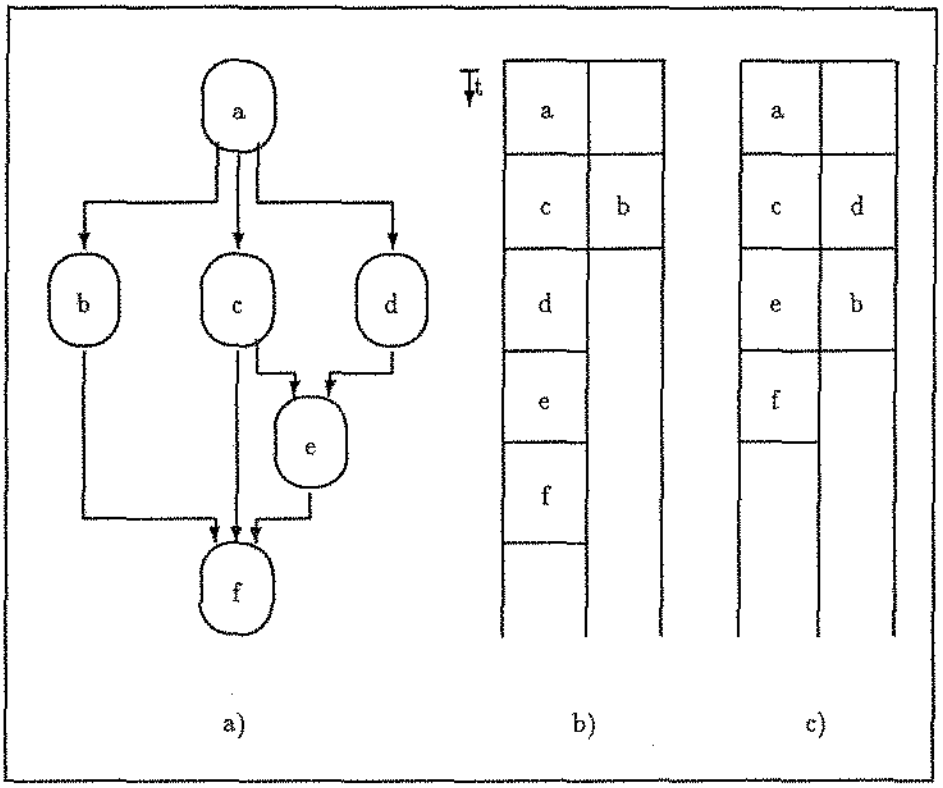

Figura 1.3: Impacto do escalonamento: a) GFD de um trecho de programa, b) *Gantt chart* do princípio *eager evaluation* e c) *Gantt chart* de outro escalonamento.

A MFDM consegue altos índices de paralelismo atuando no nível das instruções, como propõe o MFD. Contudo, como o fluxo pelo anel é seqüencial, a ordem em que os dados percorrem o sistema tem um impacto direto no desempenho da máquina. A ordem dos dados no anel define o uso dos recursos das unidades. Ela define também a habilitação das instruções e a ordem das execuções. Por estas razões, o problema de ordenação do fluxo de dados na MFDM será considerado como sendo o problema do escalonamento de tarefas da máquina.

Ao contrário do esperado, a MFDM apresenta uma perda de desempenho nas suas execuções. Também se observa uma inesperada super-utilização do dispositivo de armazenamento da unidade responsável pelo emparelhamento dos operandos que se destinam à mesma instrução. Devido a isto, esta unidade é

considerada o problema do anel e a principal causa da perda de desempenho do sistema [GW83, WG82].

1.6 Proposta

No processo de avaliação do desempenho da MFDM, duas medidas de caracterização dos programas, o paralelismo médio do programa e a proporção de operações *bypass* (ver 2.1.4), foram adotadas como medidas básicas. Com base na variação dessas medidas, o funcionamento do sistema tem sido analisado. Contudo, acredita-se que essas medidas não sejam suficientemente representativas para este tipo de uso, o que pode ter prejudicado as análises da máquina.

Nesta dissertação faz-se a reavaliação dessas medidas e adicionalmente, são identificadas as causas da perda de desempenho do sistema. Pretende-se mostrar que nem sempre a unidade de emparelhamento é uma delas.

Acredita-se também que a especialização da técnica de escalonamento adotada pela MFDM pode melhorar o desempenho do sistema. Pretende-se provar que o escalonamento das tarefas tem um impacto direto no funcionamento da máquina e que a atual política de escalonamento é uma causa indireta da perda de desempenho. Para isto, compara-se o comportamento do sistema com outras técnicas de escalonamento. Dentre estas, é analisada a CP/MISF/∞, desenvolvida neste estudo com base na filosofia de execução da MFDM com ênfase no tratamento de laços.

Como esta dissertação é um dos primeiros trabalhos do grupo da UNICAMP nessa linha, procura-se apresentar o maior número possível de referências. Assim, trabalhos futuros terão maior facilidade para seguir diferentes caminhos e buscar outras opções.

1.7 Organização do Texto

Os capítulos seguintes desta dissertação estão organizados como descrito a seguir.

O capítulo 2 resume as principais informações sobre o projeto de Manchester. A seção 2.1 descreve a MFDM, atendo-se mais detalhadamente ao funcionamento das unidades de emparelhamento e de execução. A seção 2.2 trata do simulador da máquina desenvolvido para os propósitos desta dissertação.

O capítulo 3 procura as causas reais da perda de desempenho apresentada pela MFDM. A seção 3.1 analisa a validade das medidas comumente utilizadas na avaliação das execuções. As seções 3.2 e 3.3 discutem a relação entre o funcionamento do sistema e, respectivamente, a unidade de emparelhamento e a

técnica de escalonamento de tarefas. E a seção 3.4, com base nas anteriores, define o porque da aparente perda de desempenho.

O capítulo 4 fornece uma visão geral do escalonamento de tarefas. A seção 4.1 descreve formalmente o problema, enquanto a seção 4.2 descreve alguns dos seus aspectos. A seção 4.3 analisa a importância da consideração dos custos de comunicação no escalonamento de tarefas, a seção 4.4 apresenta algumas técnicas de escalonamento e a seção 4.5 descreve a técnica CP/MISF/∞.

O capítulo 5 analisa o impacto da mudança do escalonamento de tarefas na MFDM. A seção 5.1 descreve os critérios de seleção das técnicas de escalonamento testadas. A seção 5.2 descreve como as técnicas selecionadas são integradas ao simulador. A seção 5.3 analisa o impacto dessas técnicas no funcionamento da MFDM. E a seção 5.4 analisa o impacto de CP/MISF/∞.

O capítulo 6 encerra esta dissertação. A seção 6.1 resume as conclusões. A seção 6.2 destaca as principais contribuições. E a seção 6.3 descreve algumas possíveis linhas de trabalhos futuros.

O apêndice A contém um resumido guia do usuário do simulador da MFDM desenvolvido durante este estudo.

O apêndice B contém um resumido guia do usuário do gerador de Programas-Constantes. Ele foi desenvolvido para gerar uma classe de programas utilizados durante o processo de análise das causas da perda de desempenho da MFDM.

Capítulo 2

Máquina de Fluxo de Dados de Manchester

"Quanto mais um sujeito se acha importante,
mais fácil tomar o lugar dele."

"Quando o destino fechar uma porta,
entre pela janela."

Desde os primeiros computadores von Neumann, a capacidade de computação vem crescendo rapidamente, embora sem conseguir acompanhar a demanda. Nos últimos anos, tem sido cada vez mais difícil obter ganhos consideráveis no poder computacional, existindo um movimento para o processamento paralelo [Sar89].

Uma das opções desse movimento tem sido o MFD (ver 1.4). Este possui uma simples mas poderosa capacidade de expressar o paralelismo dos programas. Há um grande número de pesquisas e publicações retratando tentativas de implementação de arquiteturas eficientes baseadas neste modelo [Pap91, HCAA93, KSY92, YSY90, OKSY92, GS92].

Um dos projetos pioneiros foi desenvolvido na Universidade de Manchester, Inglaterra. Ele teve início por volta do ano de 1976 [GW83, Vee86, Kir90] e foi idealizado e coordenado por John Gurd e Ian Watson.

Em 1978, teve início a construção do protótipo da MFDM [WG82]. E, em outubro de 1981, foi executado o primeiro programa [GKW85, WG82, Vee86]. O protótipo esteve em funcionamento até o verão de 1989 [GS92, Kir90].

O protótipo da MFDM atingiu vários de seus objetivos [Kir90] e contribuiu enormemente para o desenvolvimento de várias pesquisas na área. Contudo, seu

projeto inicial apresentou alguns problemas e limitações. Alguns aperfeiçoamentos foram implementados durante o período de funcionamento do protótipo, como o acréscimo da Unidade de Controle de Paralelismo[1] (UCP) [RS87]. Em 1992, foi publicado um relatório de progressos do projeto de Manchester [GS92], mostrando que o projeto continua ativo e evoluindo.

A seguir é descrita a MFDM. São enfatizados os pontos de sua arquitetura mais importantes para a compreensão da abordagem desta dissertação. Para a obtenção de mais detalhes, aconselha-se a consulta das referências citadas durante o texto. Por fim, descreve-se detalhadamente o simulador implementado no decurso das pesquisas, para avaliar e validar os problemas e soluções apresentados nos capítulos que seguem.

2.1 Descrição da Máquina

A MFDM [WG82, GKW85, Gur85] é uma arquitetura de fluxo de dados dinâmica [TBH82], o que permite o compartilhamento de código entre iterações de um comando repetitivo ou entre chamadas de uma mesma função. Suas tarefas têm tamanho[2] fixo e mínimo: uma instrução. As instruções têm no máximo dois operandos e produzem até dois resultados. Os dados são representados por fichas[3], compostas basicamente pelo valor do dado, um rótulo e o endereço da instrução destino. O rótulo da ficha permite a reentrância de código neste modelo [BG90b, TBH82]. As fichas são encapsuladas em pacotes e enviadas de uma unidade à seguinte.

A estrutura básica da máquina (ver Figura 2.1) é um anel com quatro unidades principais. O anel se comunica com um Sistema Hospedeiro[4] (H) através da Unidade de Chaveamento[5] (UC). Esta gerencia a saída de resultados e a entrada de novos dados no anel. As principais unidades do anel são descritas a seguir:

- Unidade de Regulagem[6] (UR) armazena temporariamente os pacotes numa fila com o propósito de manter o equilíbrio entre a produção e o consumo de fichas no anel.
- Unidade de Emparelhamento[7] (UE) sincroniza duas fichas que possuem o

[1] *Throttle Unit.*
[2] Granularidade.
[3] *Token.*
[4] *Host.*
[5] *Switching Unit.*
[6] *Token Queue.*
[7] *Matching Unit.*

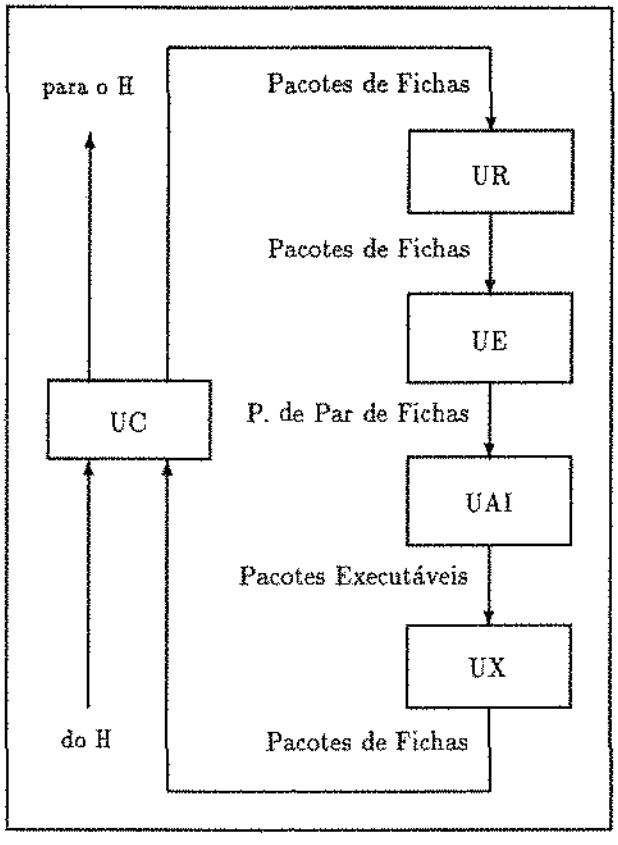

Figura 2.1: Estrutura básica da Máquina de Fluxo de Dados de Manchester.

mesmo rótulo e endereço, pois elas representam operandos da mesma instrução. A primeira ficha a chegar é armazenada enquanto espera a chegada do seu par. Após o encontro das fichas, a UE as envia num pacote de par de fichas para a próxima unidade. Fichas endereçadas a instruções de um único operando dispensam sincronização e são imediatamente enviadas à próxima unidade.

Uma característica da MFDM é sua tendência à produção de uma grande quantidade de fichas que esperam para serem emparelhadas. Este número não raramente exige a utilização de uma memória auxiliar, denominada de Unidade de Transbordo[8] (UT). Esta unidade tem tempos de manipulação

[8] *Overflow Unit.*

de dados muito mais lentos do que a UE. Assim, o seu uso deve ser evitado.

- Unidade de Armazenamento de Instruções[9] (UAI) armazena o programa representado por um GFD. A chegada do pacote de par de fichas causa o envio de um pacote executável, o qual contém a instrução e seus operandos, rótulo e endereços de destino dos resultados, para a próxima unidade.

- Unidade de eXecução[10] (UX) é composta de um conjunto de UPs. Ela recebe os pacotes executáveis e produz novos pacotes de fichas com os resultados das execuções.

As unidades da MFDM operam independentemente uma da outra num modelo *pipeline*[11] e se comunicam através de uma estrutura de fila: o primeiro pacote a chegar é o primeiro a ser processado. Todas as unidades, exceto a UX, operam seqüencialmente.

2.1.1 Partição e Escalonamento

Para que a comunicação de uma instrução com a seguinte se complete, é necessário que uma ficha circule pelo anel do sistema. Comparando-se os parâmetros da MFDM, observa-se que o tempo de processamento de uma instrução é muitas vezes menor que o tempo gasto num ciclo completo no anel [LC92]. Por isto, dever-se-ia evitar a comunicação entre instruções via o anel. Num trecho seqüencial, por exemplo, como não há paralelismo a ser extraído, seria interessante empacotar todas as instruções numa única tarefa. Isto faria com que as comunicações intermediárias fossem realizadas dentro da UX e não mais via o anel [VC92]. Contudo, a MFDM não faz tratamento especial de trechos seqüenciais, nem de qualquer outro tipo de trecho, objetivando um paralelismo total.

Na prática, observa-se que os atrasos existentes na comunicação entre instruções são muito altos. Dada a granularidade da MFDM, esta comunicação é toda realizada via o anel. Assim, muitas vezes um dado tem que transitar pelo anel para ativar a execução da instrução imediatamente seguinte (ver Figura 2.2(a)). Esta comunicação poderia se realizar internamente à UX, num tempo muito menor, caso fosse possível uma granularidade variável (ver Figura 2.2(b)).

A opção de uso de granularidade tão fina pelo projeto da MFDM segue a linha de pensamento de que a paralelização total corresponderia ao máximo de desempenho. Contudo, observa-se que nada é mais eficiente, para a execução

[9] *Instruction Store Unit.*
[10] *Execution Unit.*
[11] *Pipelined fashion.*

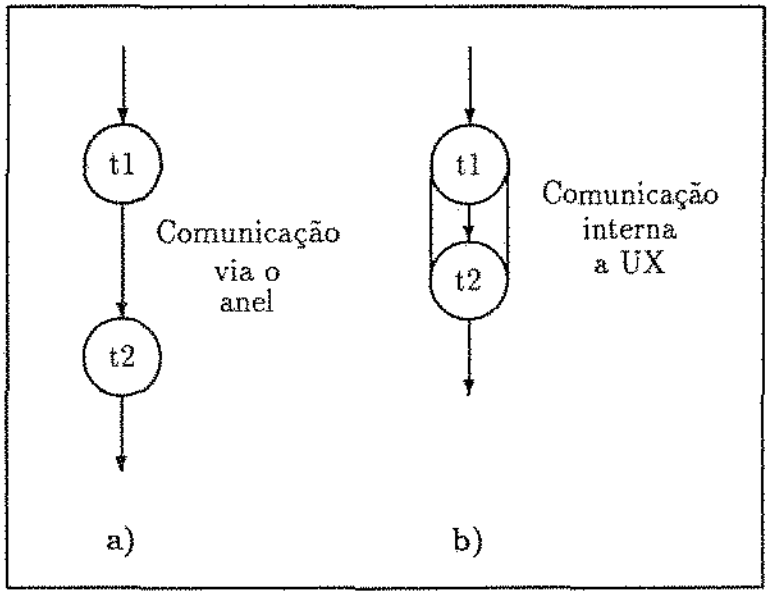

Figura 2.2: Tipos de comunicação entre instruções: comunicação a) via o anel e b) interna à UX.

de trechos intrinsecamente seqüenciais, do que o processamento explicitamente seqüencial. Além disso, devido aos atrasos de comunicação, em certo casos, mesmo trechos potencialmente paralelos podem ser mais eficientemente executados de modo seqüencial.[VC92, LC92](ver 4.3).

Na utilização do princípio de *eager evaluation*, consider-se a disponibilidade de infinitas UPs. Enquanto se considera o modelo, como não há disputa por recursos, é indiferente o tipo de política de escalonamento de instruções utilizado. Entretanto, na implementação, há um número limitado de UPs disponíveis, além de outras restrições físicas. Assim, é necessário dedicar uma atenção especial ao escalonamento de tarefas.

Como o problema de escalonamento de tarefas no MFD inclui o problema de ordenação do fluxo dos dados e como o tamanho das tarefas na MFDM é de uma instrução, considera-se instrução como sinônimo de tarefa, neste contexto. Desta mesma forma, como os dados estão encapsulados em pacotes, o termo pacote também passa a ser sinônimo de tarefa.

A MFDM utiliza a técnica de escalonamento baseada em uma lista[12], na qual as fichas são dispostas segundo alguma relação de prioridade, no caso a ordem de chegada. Esta prioridade define a ordem de execução das instruções. No contexto desta dissertação, a política de escalonamento de instruções original da MFDM é denominado FIFO[13]. FIFO não faz nenhum tipo de análise ou consideração a respeito das dependências entre as instruções existentes no grafo do programa. Assim, instruções cujos resultados somente poderão ser utilizados muitos ciclos mais tarde são muitas vezes executadas antes de outras cujos resultados são necessários imediatamente. Nesta dinâmica, muitas fichas têm que esperar por seus pares vários ciclos. Isto causa uma utilização, muitas vezes, desnecessária da memória da UE. E, nos piores casos, a utilização da UT.

2.1.2 Unidade de Emparelhamento

O que circula no anel entre as unidades de execução e de emparelhamento são pacotes de fichas (dados). As fichas podem conter um paralelismo implícito, pois, quando duas fichas são destinadas à mesma instrução, elas têm que ser emparelhadas. Esta operação ocorre na UE. Para tanto, esta unidade possui várias operações para manipulação de fichas, das quais três são essenciais[14] (ver Figura 2.3):

- *wait* – Esta operação destina-se a fichas que são argumentos de instruções com dois operandos (*Dyadic operators*[15]). Ela consistui-se da busca sem sucesso do par de uma ficha na memória da UE e pelo subseqüente armazenamento desta ficha.

  A operação *wait* envolve a manipulação do dispositivo de armazenamento da UE e de todos os seus custos. Por isto, esta operação é "lenta" para o sistema. Assim, ela é uma operação delicada, pois não repassa pacotes ao anel, embora consuma um tempo considerável. Entretanto, é necessário uma operação *wait* para viabilizar uma subseqüente operação *match*.

- *match* – Esta operação constitui-se da busca com sucesso do par de uma ficha e pelo seu subseqüente emparelhamento. Ela tem custos da mesma ordem que a operaç ao *wait*. Contudo, ela resulta na produção de um pacote, o que vai liberar uma instrução para execução.

[12] *List schedule*.

[13] *First-In-First-Out*.

[14] Para maiores detalhes sobre o conjunto de operações da UE, recomendam-se as referências [Cat81, BG90b].

[15] Na MFDM, o número máximo de operandos por instrução é dois.

- *bypass* – Esta operação é destinada exclusivamente a fichas destinadas a instruções com apenas um operando (*Monadic operators*). Ela é rápida, pois não envolve manipulação do dispositivo de armazenamento da UE. Nesse caso, a UE funciona como uma simples "porta" por onde a ficha tem que passar.

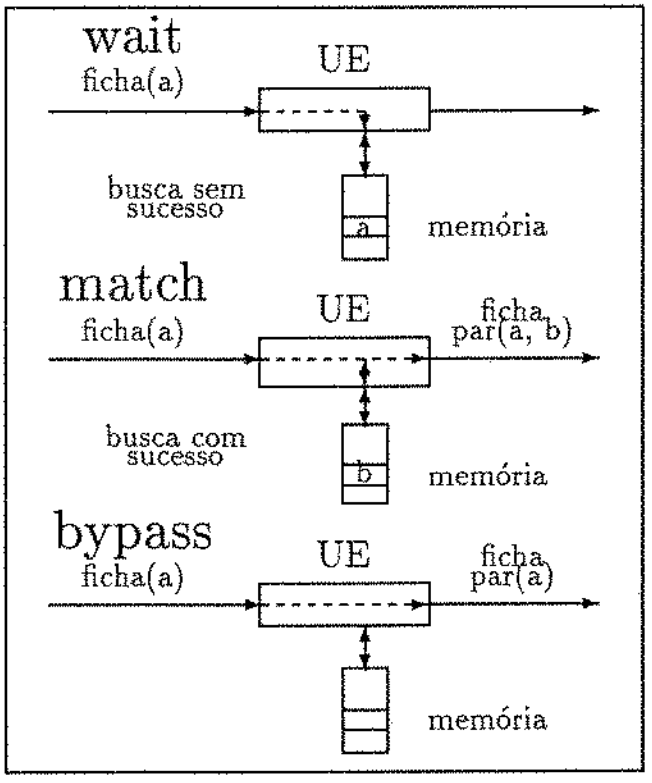

Figura 2.3: Ilustração das mais importantes operações da UE.

Dada a essencialidade e predominância dessas três operações, a maioria das análises da MFDM desconsidera as demais. Essa hipótese também será adotada neste trabalho.

Assim o comportamento da UE varia dependendo do tipo de operação efetuada. Se for *bypass*, ela é rápida e produz um pacote. Se for *match*, é lenta e também produz um pacote. Mas se for *wait*, é lenta e não há produção de pacote.

A relação entre as ocorrências destas operações é que definirá a vazão da UE e, conseqüentemente, o ritmo com que ela oferece pacotes ao sistema.

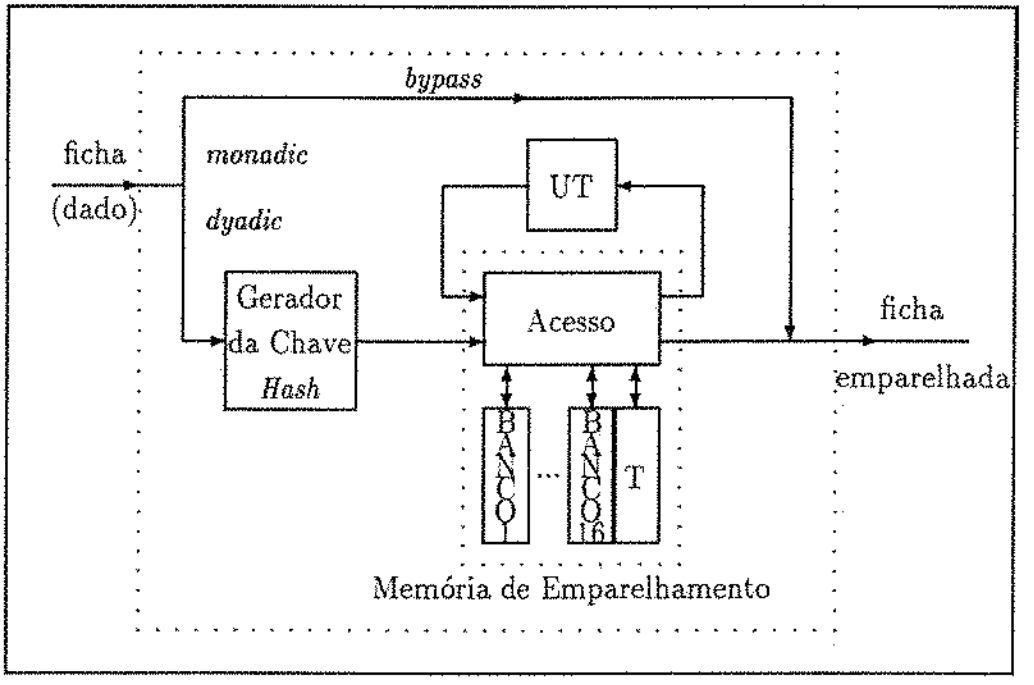

Figura 2.4: Visão interna da Unidade de Emparelhamento da MFDM.

A Figura 2.4 fornece uma visão da estrutura interna da UE [Vee86]. A UE é constituída basicamente por um conjunto de bancos de memória aos quais se tem acesso via uma função de *hash*. Quando o endereço gerado para um dado estiver ocupado em todos os bancos, o dado é encaminhado para a UT. Os dados que efetuam *bypass* são diretamente enviados para a UAI, sem passarem pela busca nos bancos de memória.

2.1.3 Unidade de Execução

A UX pode ser dividida em três partes (ver Figura 2.5): árbitro de entrada, conjunto de UPs e árbitro de saída. Esta unidade recebe como entrada pacotes executáveis e produz pacotes de fichas com os resultados das execuções.

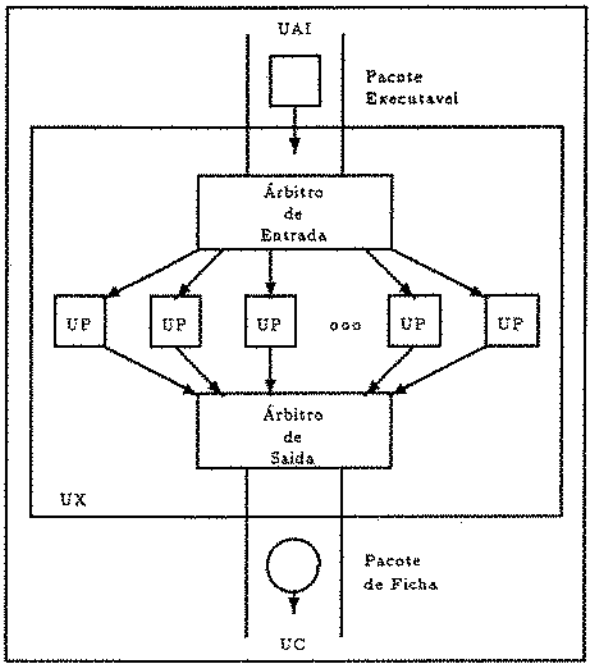

Figura 2.5: Visão interna da Unidade de Execução da MFDM.

O árbitro de entrada define a UP à qual será alocado o pacote de entrada. As UPs são idênticas e o seu número determina quantos pacotes podem ser executados em paralelo. Elas são síncronas e seu relógio tem como ciclo o tempo de execução da instrução mais demorada.

Uma UP só se torna disponível para uma nova instrução quando todas as fichas produzidas pela instrução anterior tiverem sido despachadas para o anel. Contudo, isto pode levar vários ciclos, dependendo do número de UPs que estão partilhando da seleção do árbitro de saída. Por exemplo, supondo existirem 20 UPs e que todas estejam com instruções que produzem dois resultados, a primeira UP só será liberada após vinte e dois ciclos: um ciclo para o processamento, um ciclo para liberar a primeira ficha de resultado, dezenove ciclos de espera (enquanto as outras 19 unidades estiverem liberando sua primeira ficha) e um ciclo para liberar a segunda ficha.

O árbitro de saída define qual UP terá uma ficha colocada no anel. Para isto ele usa a técnica *round robin* de modo a não ficar monopolizado por uma única unidade.

2.1.4 Medidas de Análise

Uma importante contribuição do projeto de Manchester é o conjunto de resultados das análises de desempenho de um dos primeiros sistemas baseados no MFD. Por ter sido um dos primeiros grupos a desenvolver simulações e um protótipo, o projeto teve que conduzir a avaliação do comportamento da máquina sem que existissem precedentes. Análises como a do impacto da variação do número de UPs no *speedup* e a da efetiva utilização do potencial computacional do sistema foram conduzidas na avaliação da MFDM [GW83, Das89].

Nesse processo, certas medidas foram utilizadas como base de medida. A seguir são listadas as mais importantes para o âmbito desta dissertação, juntamente com as definições aqui utilizadas:

- S1 – Número mínimo de ciclos de simulação necessários para a execução completa do programa quando apenas uma UP está disponível. Como cada tarefa corresponde a uma instrução (ver 2.1), S1 é igual ao número de instruções executadas.

- Sp – Número mínimo de ciclos de simulação necessários para a execução completa do programa quando p UPs estão disponíveis.

- Sinf – Número mínimo de ciclos de simulação necessários para a execução completa do programa quando infinitas UPs estão disponíveis. Sinf corresponde ao número mínimo de ciclos necessários para a execução de um programa.

- Pmax – Paralelismo máximo observado na execução de um programa quando infinitas UPs estão disponíveis. É iguall ao número máximo de instruções executadas simultaneamente em qualquer instante durante a execução de um programa quando infinitas UPs estão disponíveis.

- PM – Paralelismo Médio observado na execução de um programa quando infinitas UPs estão disponíveis. PM corresponde à razão de S1 por Sinf: $PM = \frac{S1}{Sinf}$.

- PMp – Paralelismo Médio observado na execução de um programa quando p UPs estão disponíveis. PMp corresponde à razão de S1 por Sp: $PMp = \frac{S1}{Sp}$.

- Tp – Tempo total de execução de um programa quando p UPs estão disponíveis.

- Oby – Número de operações *bypass* realizadas na UE na execução de um programa. Oby é igual ao número de instruções de apenas um operando executadas pelo programa.

- Ow – Número de operações *wait* realizadas na UE na execução de um programa. Considerando-se apenas operações *bypass*, *wait* e *match* (ver 2.1.2), Ow é igual ao número de instruções de 2 operandos executadas pelo programa.

- Om – Número de operações *match* realizadas na UE na execução de um programa. Considerando-se apenas operações *bypass*, *wait* e *match* (ver 2.1.2), Om é igual ao número de instruções de 2 operandos executadas pelo programa.

- Pby – Proporção das operações *bypass* pelo número total de operações realizadas na UE. Destaca-se que, como descrito em 2.1.2, existem outros tipos de operações realizadas pela UE, mas elas não são consideradas nesta dissertação. Assim sendo, $Pby = \frac{Oby}{Oby+Ow+Om}$. Como cada instrução de dois operandos gera uma operação *wait* e uma *match*, Ow e Om devem possuir o mesmo valor, igual ao número de instruções de dois operandos executadas pelo programa.

As medidas S1, Sinf, Pmax, PM, Oby, Ow, Om e Pby são características do programa, independentes das características da máquina. Já as medidas Sp, PMp e Tp são resultados do comportamento do programa sob a restrição do número de UPs disponíveis, p. E, sendo $p < Pmax$, o programa sofre a influência de outras restrições do sistema, como, p. ex., a política de escalonamento de instruções.

2.1.5 Alguns Problemas em Aberto

No decorrer do desenvolvimento da MFDM, alguns problemas e limitações foram observados, dentre os quais destacam-se:

- Tratamento de Estruturas de Dados

  Pode-se classificar o tratamento de estruturas de dados como um problema geral às implementações do MFD. A passagem de estruturas completas de uma instrução à outra se apresenta inviável pelo seu alto custo. Por outro lado, o uso de ponteiros para uma unidade de armazenamento de estruturas de dados quebra a elegância formal do paradigma [Das89]. Este problema tem recebido grande atenção e várias soluções foram oferecidas [Kam94].

A MFDM original não trata estruturas de dados especificamente, mas este tema foi abordado em versões posteriores [GS92, Kam94].

- Emparelhamento de Dados

  O emparelhamento das fichas de dados é um dos problemas mais complicados em máquinas de fluxo de dados dinâmicas. As soluções são praticamente tantas quanto as propostas de implementação do modelo (para exemplos, ver [CC93]). A MDFM utiliza uma memória pseudo-associativa, onde uma técnica de *hashing* por *hardware* é utilizada [Cru]. Em [CC93, Cru] é proposta uma reformulação da arquitetura original da MFDM, onde este procedimento é alterado.

- Granularidade das Tarefas

  Como visto em 2.1, a MFDM explora o paralelismo do programa a nível das instruções. Esta conduta visa paralelizar o que for possível nas execuções. Contudo, como descrito em 1.4.1, ela pode introduzir atrasos na comunicação entre as instruções. Neste caso, ao invés de ocorrer o aumento do *throughput* do sistema, pode ocorrer a sua diminuição [LC92].

- Controle do Paralelismo

  Muitas aplicações, por algum tempo, apresentam excesso de paralelismo [RS87], o que acaba por provocar a super-utilização dos dispositivos de armazenamento do sistema, principalmente a memória da UE. Na tentativa de controlar o paralelismo disponível, foi idealizada a UCP [RS87, GS92]. Com objetivo similar, esta dissertação aborda o controle de paralelismo através do escalonamento das tarefas, o que elimina a necessidade da adição de mais uma unidade ao anel.

2.2 Descrição do Simulador: gMDMS

A MFDM é um anel por onde os dados circulam seqüencialmente. Na UR, os dados aguardam seqüencialmente para serem enviados à UE. Nesta, um por um, os dados são processados: ou aguardam por seu par, ou seguem "solteiros" ou "casados" para a UAI. Os dados "solteiros" e os "casados", ao passarem pela UAI, "vestem" as instruções a que se destinam e seguem, em fila, para a UX. O árbitro de entrada da UX aloca cada instrução a uma UP diferente, e aí, não mais seqüencialmente, as instruções são processadas. O árbitro de saída coordena o

envio seqüencial dos dados à UC. Esta desvia os dados finais para o H e os dados intermediários para a UR, fechando o ciclo do anel.

O objetivo do simulador descrito a seguir é representar a execução da máquina de Manchester em termos do fluxo e da ordem das fichas de dados, pois, para o propósito de escalonamento, interessa estudar o efeito da seleção sobre a ordem de execução das instruções, que, no caso, é determinada pela ordem dos dados. Não há necessidade de reconhecer o valor dos dados ou o tipo de operação aritmética implementada por uma instrução. Tem-se necessidade de reconhecer apenas as instruções de controle e o aspecto do grafo do programa. É o mesmo que trabalhar na abstração que representa os dados como "bolas pretas" fluindo pelos "canos" do programa (ver Figuras 1.1 e 1.2).

O simulador desenvolvido para a aplicação nesta dissertação é denominado gMDMS, Gifu - Manchester Dataflow Machine Simulator. Ele foi implementado durante a permanência do autor na Universidade de Gifu, Gifu, Japão. gMDMS é escrito na linguagem C e se encontra na versão 3.1. O *layout* dos arquivos de resultados é compatível com o *software* de geração de gráficos GNUPLOT.

O Apêndice A contém um resumido Guia do Usuário do gMDMS. Suas principais características, decisões de projeto e restrições são descritas a seguir.

2.2.1 Modelagem

Como descrito em 2.1, a MFDM é formada por um conjunto de unidades conectadas uma a uma por um canal seqüencial. Essas conexões formam um anel com acesso a um sistema exterior através de uma das unidades.

Todas as unidades, exceto a UX, trabalham de forma seqüencial. Nelas, apenas um pacote é processado por vez. Na UX, os pacotes entram e saem também um por vez, mas várias instruções podem ser executadas em paralelo. O grau máximo de paralelismo explorável na UX e, conseqüentemente na máquina, é igual ao número de UPs existentes.

Considerando-se que a UE só execute as operações descritas em 2.1.2, a ordem com que os dados saem da UX é mantida durante o percurso de retorno a ela. Os únicos pontos onde essa ordem pode ser alterada são a UC e a UE. Na UC, os dados finais são encaminhados para fora do anel, H. E na UE, os primeiros dados endereçados a instruções de duas entradas são forçados a esperar pelos seus pares.

Como em geral a UC só é utilizada no início da execução, entrada dos dados iniciais, e no final, saída dos resultados finais, ela não é representada na modelagem do gMDMS[16]. Esta simplificação restringe as simulações a programas que

[16] A identificação do final da execução de um programa é realizada por sinalizadores inseridos

só se comunicam com o H no início e no final da execução.

A Figura 2.6 mostra a estrutura geral do gMDMS. As unidades de execução, de emparelhamento e de armazenamento de instruções são vistas como centrais de serviços com uma fila associada. A UR, por ser também uma fila, é combinada à fila da UE. Por fins de clareza, as filas das unidades de emparelhamento, armazenamento de instruções e execução serão referenciadas respectivamente por fUE, fUAI e fUX. Modelagem semelhante foi utilizada por D. Ghoshal e L. N. Bhuyan [GB87]. O trabalho de Ghoshal e Bhuyan é referenciado e analisado por L. Dasgupta em [Das89]. Os trabalhos [GW83, SC92] são exemplos de outras modelagens de simulação da MFDM.

As unidades da MFDM são assíncronas. Contudo, como elas estão conectadas entre si, o consumo de uma depende da produção da anterior. Assim, a unidade mais lenta domina a velocidade do fluxo de todo o anel. A seguir são descritos a modelagem de cada unidade e as relações temporais entre elas.

2.2.2 Representação das Unidades

Várias simplificações foram realizadas na implementação deste simulador em relação à MFDM. Contudo, nenhuma delas desvirtua o funcionamento original da máquina. Estas simplificações, descritas a seguir, foram possíveis devido às necessidades específicas deste trabalho.

O fato de não ser necessário obter valores para os resultados das execuções é certamente uma das principais concessões à implementação do simulador. Como não há valores, não é necessário:

- executar as instruções – Basta contabilizar o tempo que a execução levaria, analisar o efeito das instruções de controle e enviar um sinal às instruções seguintes de que os resultados estão disponíveis.
- armazenar os dados que esperam por seus pares na UE – Basta armazenar um sinal representando a existência do dado. Isto possibilita grandes simplificações na UE.

Durante a descrição da implementação das unidades, outras simplificações específicas serão mencionadas.

Modelagem da UE

Na execução de um grafo por uma arquitetura de fluxo de dados dinâmica, podem existir mais de um dado em cada aresta esperando por seu par, para poder

na implementação do simulador.

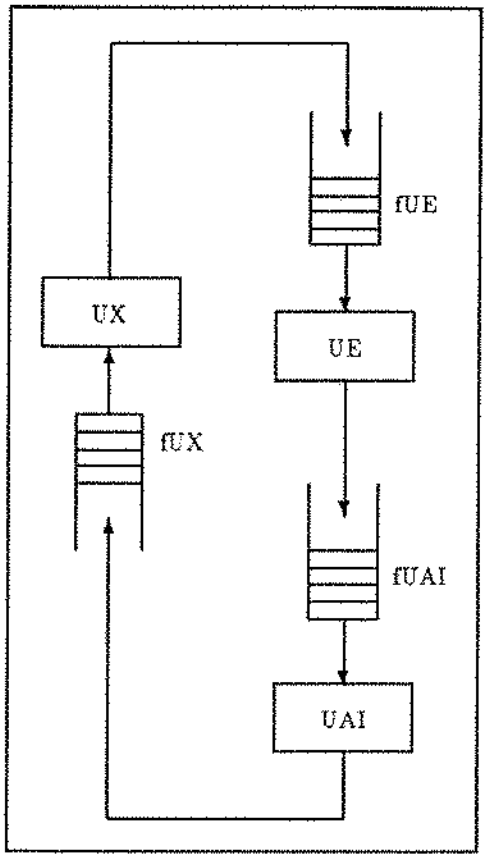

Figura 2.6: Estrutura geral do gMDMS.

habilitar a instrução à execução (ver 1.4). Em uma instrução de dois operandos, supondo-se que a ordem com que os dados chegam seja a mesma nas duas arestas de entrada, em apenas uma delas poderá formar-se uma fila. Assim, sempre que existirem dados em ambas as arestas, eles serão pares. Esta consideração não é verdadeira em alguns casos, como, p. ex., quando ocorrem várias chamadas concomitantes de uma mesma função.

Por simplicidade, a versão atual de gMDMS considera que a ordem de chegada dos dados é a mesma em todas as arestas. Mesmo não sendo uma consideração genérica, ela possibilita grandes simplificações sem restringir grandemente as classes de programas analisados.

Levando em conta essas considerações, a memória da UE pode ser representada por um conjunto de contadores e sinalizadores. Cada instrução está associada a um contador que armazena o número de fichas de dados que estão esperando por seu par. Também, para cada instrução, existe um sinalizador que indica, para cada ficha esperando, a aresta em que ela se encontra. Assim, a representação da memória da UE fica reduzida a um espaço proporcional ao número de instruções existentes no programa.

Também o método de pesquisa da posição da ficha na memória é simplificado. Ao invés de usar *hashing*, usa-se acesso direto através da identificação da instrução endereçada. Isto é viável pois o grafo do programa é estático, não se alterando durante a execução [Vee86]. Em gMDMS, o grafo do programa é representado por um vetor de instruções.

Como a representação da memória da UE é um reflexo do grafo do programa, este foi representado, por conveniência, na UE. Esta decisão não afasta o simulador da máquina, pois os tempos de cada unidade são respeitados. Assim sendo, o pacote que sai da UE já leva consigo a instrução a ser executada. Ao pacote passar pela UAI, o tempo gasto na máquina é adicionado ao relógio de simulação (ver 2.2.3).

O funcionamento da UE fica, então, como descrito a seguir. A ficha de dado é lida na fUE e sua instrução destino é buscada na memória da UE. Caso a instrução tenha apenas um operando, um pacote é enviado para a fUAI (operação *bypass*). Caso a instrução possua dois operandos, verifica-se, através do sinalizador de ocupação, se há alguma outra ficha aguardando. Caso não exista, o sinalizador é ajustado de acordo com a entrada à qual a ficha pertence e é incrementado o contador de fichas em espera (operação *wait*). Caso exista alguma ficha aguardando par, compara-se o valor do sinalizador com o valor do operando ao qual a ficha se destina. Se forem iguais, o contador é incrementado (operação *wait*); caso sejam diferentes, o contador é decrementado e um pacote é enviado à fUAI (operação *match*). Caso o contador se torne zero, o sinalizador é ajustado para vazio, o que significa que não existem fichas de dados para essa instrução esperando ser emparelhadas.

Modelagem da UAI

A função básica da UAI na MFDM é armazenar o grafo do programa. Como em gMDMS esta função já é representada na UE, a UAI figura apenas como um módulo de atraso no envio dos pacotes executáveis à UX. A eqüivalência da UAI do simulador com a da máquina é preservada pela relação dos tempos de cada unidade (ver 2.2.3).

Modelagem da UX

Como não existem valores, não há o que ser executado, exceto pelas instruções de controle (ver 2.2.4). Como em [GW83], supõe-se que todas as instruções tenham o mesmo tempo de execução. Assim, a execução prossegue em ciclos discretos de mesma duração.

A UX é representada por um vetor de tamanho correspondente ao número de UPs disponíveis no sistema. Segundo a técnica *round robin*, adotada na MFDM (ver 2.1.3), os resultados são escoados para o anel na ordem em que se encontram nesse vetor, sendo realizadas duas varreduras, uma para os primeiros resultados e outra para os segundos. Os resultados das instruções que possuem uma única saída são escoados na primeira varredura. A ordem de alocação das instruções nas UPs segue a ordem na qual elas aparecem na fUX: o primeiro na fila ocupa a primeira posição do vetor e assim por diante, até o preenchimento das UPs disponíveis ou o esvaziamento da fila.

Modelagem das Filas

As filas de cada unidade são implementadas pela mesma estrutura de dados. A política de ordenação desta estrutura pode ser alterada, possibilitando variar a distribuição dos elementos. Como estão sendo testadas técnicas de escalonamento baseadas em listas, a política de ordenação nas filas varia de acordo com a política de ordenação empregada (ver 4.4). Na execução da política original da MFDM, a ordenação ocorre pela ordem de chegada dos pacotes. Por este motivo, emprega-se o termo fila[17] no texto.

2.2.3 Relação de Produção entre as Unidades: Tempos

É possível obter uma relação de produção entre as unidades do sistema. O número de instruções processadas em paralelo é igual ao número de UPs utilizadas. Contudo, devido a custos de alocação e outros custos de gerência dos resultados, a produção da UX em relação ao número de UPs em uso é não linear. Baseando-se em típicos resultados do protótipo de Manchester [GKW85], aqui reproduzidos na Tabela 2.1, e nas velocidades das unidades, pode-se calcular as relações de tempos "típicas" entre elas.

Na MFDM, em um segundo é possível processar:

- 1,11 $x10^6$operações *wait/match* na UE[18].

[17] *Queue.*

[18] Não se considera o uso da UT.

Tabela 2.1: Conjunto típico de resultados do tempo de execução.

# PUs (p)	Tempo de Processamento (Segundos)	*Speedup*	Eficiência	MIPS Reais (Mp)	MIPS Potenciais
1	4,4215	1,00	100,0	0,117	0,117
2	2,2106	2,00	100,0	0,235	0,235
3	1,4751	3,00	99,9	0,352	0,352
4	1,1077	3,99	99,8	0,469	0,470
5	0,8886	4,98	99,5	0,585	0,587
6	0,7429	5,95	99,2	0,699	0,705
7	0,6400	6,91	98,7	0,812	0,822
8	0,5643	7,84	97,9	0,921	0,940
9	0,5071	8,72	96,9	1,024	1,057
10	0,4629	9,55	95,5	1,122	1,175
11	0,4301	10,28	93,5	1,208	1,292
12	0,4038	10,95	91,3	1,287	1,410

- 5,56 $x10^6$ operações *bypass* na UE.
- 2,00 $x10^6$ *fetches* de instruções na UAI.

Adotando-se que:

- p representa o número de UPs em uso. Primeira coluna da Tabela 2.1. Como a granularidade da MFDM é de uma instrução (ver 2.1) e o tempo de execução de cada instrução é suposto ser o mesmo (ver 2.2.2), a unidade de medida de p é eqüivalente à unidade de medida instrução (inst).
- Mp representa, em MIPS, a proporção do uso do conjunto de UPs, quando p estão disponíveis. Mp corresponde à razão de $S1$ por Tp: $Mp = \frac{S1}{Tp}$ [GW83]. Quinta coluna da Tabela 2.1.

Tem-se que, como a MFDM trata-se de um sistema paralelo, a razão de Mp por p, $\frac{Mp}{p}$, expressa o número de ciclos de execução que a UX realiza, ou seja, o número de instruções processadas por uma UP, em um segundo quando p UPs estão disponíveis. Logo, a razão de p por Mp, $\frac{p}{Mp}\left(\frac{\frac{inst}{10^6 inst}}{sec} \Longrightarrow \frac{sec}{10^6}\right)$, eqüivale

ao tempo necessário para se executar p instruções em paralelo, ou seja, o tempo total de um ciclo de execução, ou seja, o tempo para se executar uma instrução, quando p UPs estão disponíveis.

Tendo em mãos a relação que expressa o tempo de uma execução na UX, pode-se, então, fazer uma relação entre as produções das unidades do sistema (ver Tabela 2.2). Para tanto, basta multiplicar a razão $\frac{p}{Mp}$ pela produção de cada unidade em um segundo $\left(\frac{10^6 inst}{sec}\right)$. Como forma de simplificação, as unidades 10^6 são reduzidas. Então, tem-se que:

$$fUEwm = 1,11 * \frac{p}{Mp} \qquad fUEby = 5,56 * \frac{p}{Mp} \qquad pUAI = 2,00 * \frac{p}{Mp}$$

Onde:

- $fUEwm$ representa o número de fichas que realizam *wait/match* na UE. Se for *match* o número de pacotes produzidos é o mesmo que o número de fichas consumidas. Se for *wait*, as fichas são consumidas, mas não há produção de pacotes (ver Tabela 2.2).
- $fUEby$ representa o número de fichas que realizam *bypass* na UE. O número de pacotes produzidos é o mesmo que o número de fichas consumidas.
- $pUAI$ representa o número de pacotes processados pela UAI. O número de instruções produzidas é o mesmo que o número de pacotes consumidos.

Assim, com esses valores da relação de produção entre as unidades, pode-se sincronizar as unidades em relação à UX. Se num ciclo, 10 UPs estiverem disponíveis, p. ex., a UE pode processar entre 9,89 (*wait/match*) e 49,55 (*bypass*) pacotes disponíveis da fUE. Já a UAI pode processar 17,82 pacotes disponíveis da fUAI. Como só podem ser processados valores inteiros de pacotes, algumas compensações devem ser realizadas nos ciclos seguintes.

A relação entre os custos das operações *wait/match* e *bypass* é aproximadamente cinco. Por isso, contabiliza-se uma operação *wait/match* como sendo cinco *bypass*. Este é um ponto importante para não se incorrer no erro de, p. ex., no caso anterior, processar na UE 9,66 pacotes *bypass* e 48,39 pacotes *wait/match*.

Como os dados disponíveis nas publicações relativas à MFDM se limitam a até 12 UPs disponíveis, gMDMS realiza simulações para esse intervalo considerando os valores descritos anteriormente na relação entre as unidades. Para um número superior a 12 UPs, consider-se que as unidades têm tempo suficiente para processar todos os pacotes existentes nas suas filas. Utiliza-se a simulação com um número infinito de UPs disponíveis, o que significa que a UX pode consumir em cada ciclo todos os pacotes existentes na sua fila, para a análise das características gerais do programa, como, p. ex.: PM, Pmax e Pby (ver 2.1.4).

Tabela 2.2: Produção entre as unidades.

#UPs	UX	UE (# Fichas)			UAI (# Pacotes)
		bypass	*wait*	*match*	
1	2	47,52	0	9,48	17,09
2	4	47,31	0	9,44	17,02
3	6	47,38	0	9,46	17,04
4	8	47,42	0	9,46	17,05
5	10	47,52	0	9,48	17,09
6	12	47,72	0	9,52	17,16
7	14	47,93	0	9,56	17,24
8	16	48,29	0	9,64	17,37
9	18	48,86	0	9,75	17,57
10	20	49,55	0	9,89	17,82
11	22	50,62	0	10,10	18,21
12	24	51,84	0	10,34	18,64

2.2.4 Representação das Instruções

Como argumentado anteriormente, gMDMS não representa os valores dos resultados das instruções, as quais, seguindo a linha do simulador descrito em [GKW85], têm igual tempo de execução. Assim, é necessário apenas identificar as instruções responsáveis pelo fluxo de controle.

Em gMDMS, identificam-se somente as entradas e saídas de cada vértice do grafo do programa e os vértices que correspondem às instruções de controle (*branch*, *call*, etc). Existem nove tipos de instruções implementadas em gMDMS: *Normal*, *Branch*, *Reset*, *Consumer*, *Call*, *Begin Recursive*, *Begin Non Recursive*, *End* e *End of Program*. A Tabela 2.3 mostra o número de entradas e de saídas de cada uma delas. Com base nestas instruções são formados os grafos dos programas.

A instrução *Normal* representa genericamente as instruções que não são de controle. *Branch* tem um contador que define qual aresta de saída será usada em lugar do argumento booleano do caso real. *Consumer* elimina fichas e *Reset* atribui novos valores para os contadores das instruções *Branch* e *Consumer*. A instrução *Reset* não é considerada uma instrução executável, isto é, ela não é

Tabela 2.3: Tipos de instruções implementadas em gMDMS.

Tipo da Instrução	# Entradas	# Saídas
Normal	1, 2	1, 2
Branch	1, 2	2
Reset	1	1
Consumer	1, 2	1, 2
Call	1, 2	1
Begin Recursive	1	1, 2
Begin Non Recursive	1	1, 2
End	1, 2	1
End of Program	1, 2	1

executada na UX e, assim, não afeta o paralelismo do sistema. A instrução *Call* ativa procedimentos recursivos e não-recursivos, os quais sempre começam, respectivamente, com instruções *Begin Recursive* e *Begin Non Recursive* e sempre terminam com a instrução *End*. O programa principal (*main()*) sempre termina com a instrução *End of Program*.

Exemplos do uso dessas instruções na representação dos programas são encontrados no capítulo seguinte.

2.2.5 Validação do Simulador

gMDMS não tem por objetivo ser um simulador genérico e nem abrangente. O seu propósito é habilitar o estudo do impacto de diferentes técnicas de escalonamento de instruções na MFDM.

Nesse sentido, ele é um reflexo da análise do funcionamento lógico da MFDM. Algumas restrições foram adotadas para possibilitar simplificações na sua implementação. Contudo, as restrições existentes não impedem que ele seja uma representação válida da MFDM. Os programas considerados na simulação têm características básicas de computação que permitem a generalização dos resultados obtidos. Extensões para estudos mais específicos em partição e em escalonamento são também viáveis, mas necessitando pequenas alterações no modelo básico adotado.

A seguir, resumem-se as principais restrições impostas nos programas executados em gMDMS:

- Só há comunicação com o H no início e final da simulação. Apenas os resultados finais podem ser enviados ao H.
- Os tempos de execução de todas as instruções são considerados iguais.
- Como não existe representação dos valores dos dados, as operações de desvio são ajustadas estaticamente antes da simulação.
- A ordem com que os dados são enviados a cada uma das arestas de uma instrução de dois operandos é considerada a mesma. Assim, fica restrito o efeito de chamadas concomitantes de uma mesma função.

Ao longo da realização deste trabalho, os resultados obtidos das simulações e utilizados nesta dissertação foram sempre coerentes com aqueles disponíveis na bibliografia afim, destacando-se [GW83, GKW85, BG90b, Gur85].

Capítulo 3

Análise do Problema: Por que a Perda de Desempenho?

"Seja insaciavelmente curioso.
Pergunte sempre 'por quê?'."

"Seja mais esperto que os outros,
mas não deixe que eles saibam."

As medidas PM e Pby têm sido adotadas como parâmetros para análise da potencialidade dos programas executados na MFDM. O gráfico da Figura 3.1 mostra duas curvas de desempenho típicas – "número de UPs utilizadas" *versus* "número de UPs disponíveis" – resultantes da execução na MFDM de dois programas distintos, mas ambos com PM = 50 e Pby = 80%. Observa-se que, embora esses parâmetros sejam altos nos dois casos, existe uma perda de desempenho visível. A UE tem sido considerada a causa desta perda.

Neste capítulo faz-se uma análise das causas reais da perda de desempenho na MFDM. Para tanto, é reavaliada a eficácia de Pby e PM como indicadores. Em seguida, a UE é descrita e analisada em mais detalhes. Também é discutido o efeito da técnica de escalonamento FIFO no funcionamento da máquina. Por fim, argumenta-se "Porque a Perda de Desempenho" na MFDM.

3.1 Análise das Medidas

As medidas Pby e PM (ver 2.1.4) têm sido aceitas como bons indicadores de desempenho dos programas na MFDM. E, conseqüentemente, elas têm sido uti-

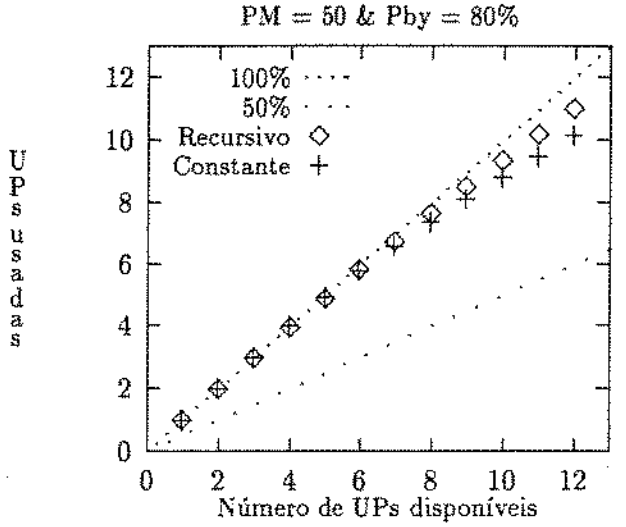

Figura 3.1: Curvas de desempenho típicas das execuções na MFDM.

lizadas na análise do funcionamento do sistema. Mesmo sendo medidas "brutas", elas são consideradas indicadores consistentes do desempenho das execuções [GW83, GKW85].

Embora exista uma aprovação pelos trabalhos afins da "qualidade" dessas duas medidas, Pby e PM, é realizada a seguir uma reavaliação delas. Acredita-se aqui que, em vez de medidas médias, as variações das operações da UE (p. ex., Figura 3.2) e do paralelismo pela execução do programa são dados mais adequadas, para a análise do desempenho da execução de programas na MFDM e do desempenho do sistema em si.

3.1.1 Proporção de *Bypass*

Com o objetivo de analisar o impacto da distribuição das operações da UE no desempenho da máquina, dois grupos de programas, chamados Programas-Constantes, foram gerados[1]. Todos os programas dos dois grupos têm uma distribuição constante do paralelismo e PM igual a 50 (S1 = 5.000 e Sinf = 100).

[1]Para tanto, foram desenvolvidos os geradores de Programas-Constantes CPG1 e CPG2 (Constant-Programs Generator). O Apêndice B contém um guia do usuário resumido destes geradores.

Isto significa que a configuração de 50 UPs disponíveis é a ótima. A execução em menos de 100 ciclos, mesmo com mais de 50 UPs disponíveis, não é possível, devido às dependências de dados entre as instruções.

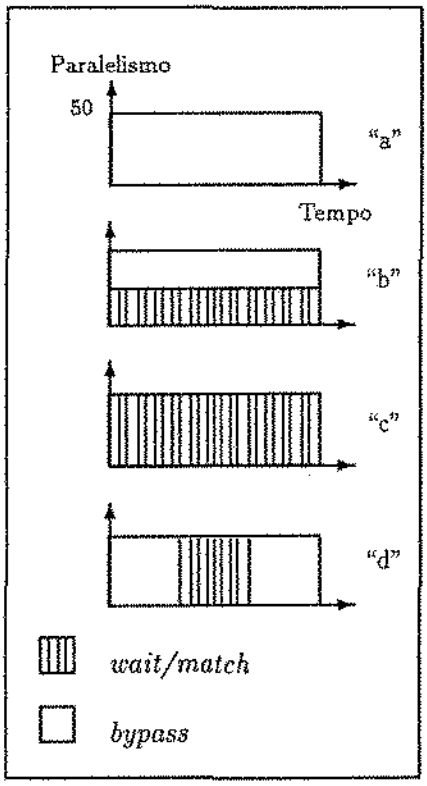

Figura 3.2: Tipos de silhueta da distribuição das operações *wait/match*.

Os programas do grupo 1 têm perfis do tipo "a", "b" ou "c" da Figura 3.2, decorrentes dos códigos mostrados nos trechos respectivos da Figura 3.3. Assim, é possível analisar os índices de *wait/match* que degradam o *throughput* da UE. Já os programas do grupo 2 têm perfis do tipo "d" da Figura 3.2, o que corresponde a uma parte do programa do tipo "c" e o restante do tipo "a" da Figura 3.3. Isto possibilita a análise do efeito de concentrações de *wait/match*.

Sete programas do grupo 1 são analisados. Neles, diferentes tamanhos da faixa (longitudinal) de *wait/match* resultam em diferentes níveis de Pby. Do grupo 2, analisam-se quinze programas, divididos em três subgrupos. Nestes, os

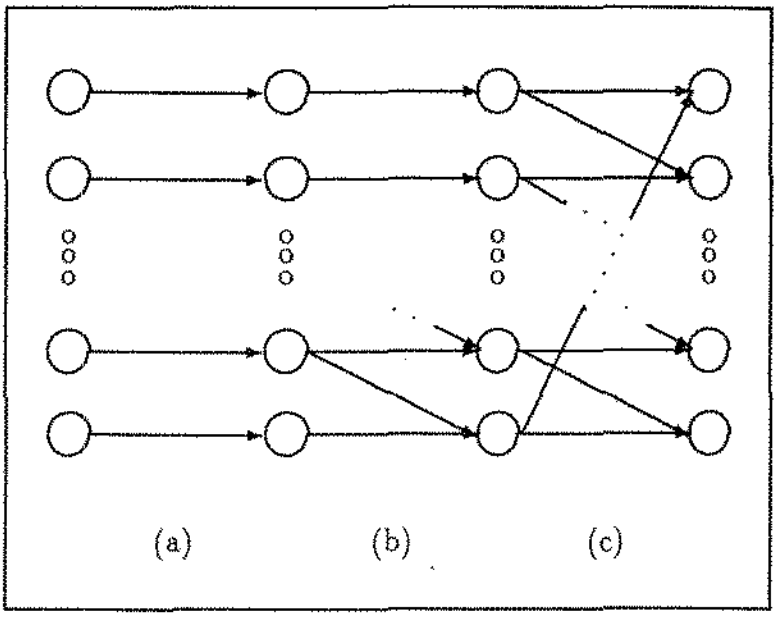

Figura 3.3: Tipos de grafos, níveis de Pby iguais a a) 100%, b) intermediários e c) 0%.

diferentes tamanhos das faixas (transversais), posicionadas no início, meio e fim do programa, provocam diferentes níveis de Pby. Nestes dois grupos, chama-se de tamanho da faixa *wait/match* o número de instruções que determina o Pby do programa. As Figuras 3.4 e 3.5 mostram exemplos da relação entre o tamanho da faixa *wait/match* e o perfil da distribuição das instruções dos programas dos grupos 1 e 2, respectivamente. Os níveis de Pby em relação ao número de instruções pertencentes às faixas *wait/match* estão relacionados na tabela 3.1.

O gráfico da Figura 3.6 contém as curvas de desempenho dos programas do grupo 1. Observa-se uma perda de desempenho nos programas cujo nível de Pby é inferior a 66%. O gráfico da Figura 3.7 contém as curvas de desempenho dos programas do grupo 2. Os programas com mesmos níveis de Pby apresentam as mesmas curvas de desempenho, independentemente do subgrupo a que pertencem. Neste grupo, observa-se uma perda de desempenho em todos os programas.

A perda de desempenho observada nos gráficos das Figuras 3.6 e 3.7 não pode

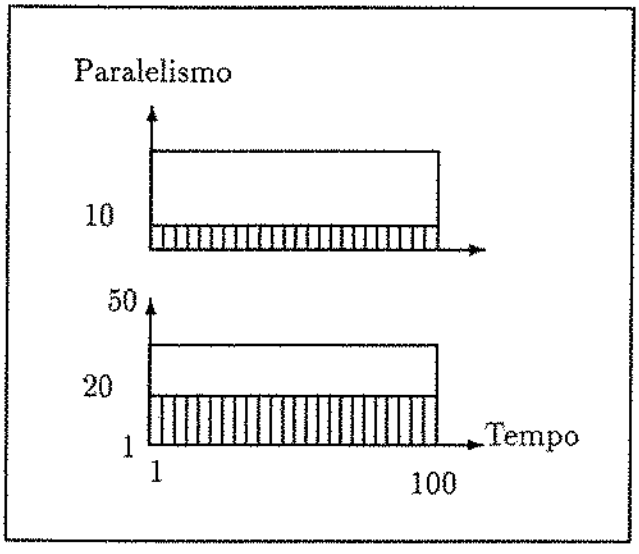

Figura 3.4: Exemplos da relação entre o tamanho da faixa *wait/match* e o perfil da distribuição das instruções dos programas do grupo 1.

Tabela 3.1: Níveis de Pby.

Tamanho da faixa *wait/match*	Pby (%)	
	Grupo 1	Grupo 2
0	100,000	–
5	81,818	90,385
10	66,666	81,651
20	42,857	66,387
30	25,000	53,488
40	11,111	42,446
50	0,000	–

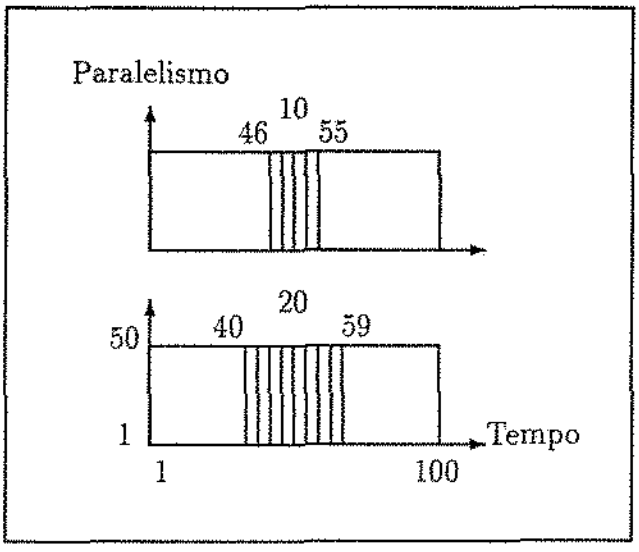

Figura 3.5: Exemplos da relação entre o tamanho da faixa *wait/match* e o perfil da distribuição das instruções dos programas do grupo 2.

ser atribuída à falta de paralelismo, pois este se apresenta com valor alto e com distribuição constante em todos os programas. Nem o nível de Pby é em si a causa, pois, comparando-se os dois gráficos, programas com níveis de Pby semelhantes apresentam curvas distintas. Então, a causa tem que ser a característica distinta entre esses programas: a distribuição das operações da UE.

Nos programas do grupo 1, a distribuição das operações da UE é uniforme durante todo o tempo de execução. Assim as operações *bypass* suavizam os efeitos das operações *wait/match* no *throughput* da UE. Contudo, nos programas do grupo 2, existe a concentração das operações *wait/match*, que reduz o *throughput* da UE. Os níveis de produção de pacotes da UE prevalece no fluxo do anel durante as faixas de concentrações das operações *wait/match*. Assim, mesmo que haja um grande o número de UPs disponíveis, elas não são utilizadas efetivamente.

A UE passa a ser um problema para o sistema, um "gargalo", quando ocorrem concentrações de operações como as *wait* e *match*. Nestes casos, é como se o sistema sofresse um "entupimento". Contudo, o problema não é expresso pela medida Pby. Mesmo quando o código possui um valor alto nesta medida, pode haver concentrações de *wait* e *match* que "entupirão" momentaneamente o sistema.

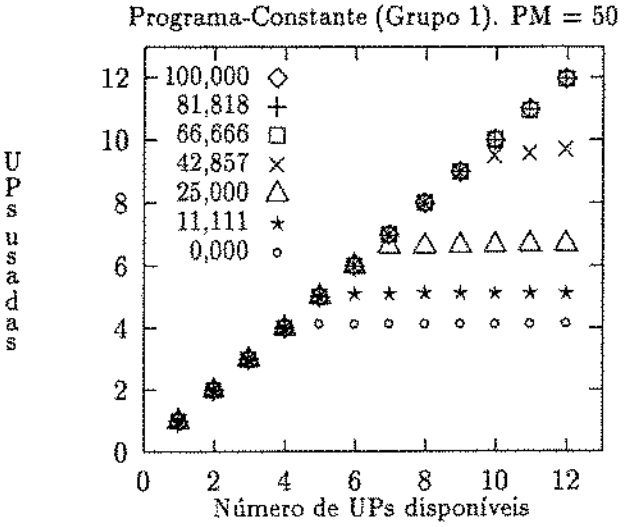

Figura 3.6: Curvas de desempenho dos programas do grupo 1, variando-se os níveis de Pby entre 0 e 100%.

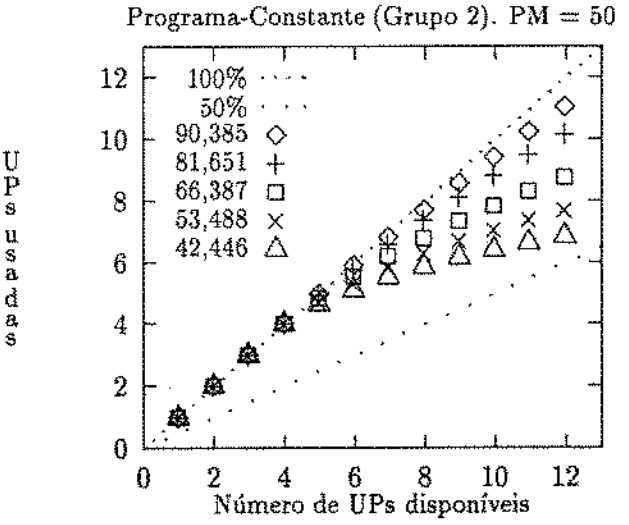

Figura 3.7: Curvas de desempenho dos programas do grupo 2, variando-se os níveis de Pby de 42 a 90%.

Esse tipo de concentração pode ser causado pelo processo de ordenação das fichas de dados no sistema ou pode ser intrínseco ao código do programa. No segundo caso, somente alterações no código podem evitar a ocorrência do problema. É preciso lembrar que a UE é muito mais rápida do que a UAI nas operações *bypass*, mas é muito mais lenta nas operações *wait* e *match*. Esta diferença faz com que, mesmo em situações onde a UE tem taxa de "escoamento" baixa ou nula, possa ainda assim haver na fila entre a UE e UAI um número suficiente de pacotes, para que o suprimento de instruções à UX não seja imediatamente afetado. Isto deve-se a diferentes histórias do comportamento do *throughput* dessas unidades.

A fila entre a UE e a UAI funciona como um "amortecedor", uma caixa d'água: mesmo que o fornecimento domiciliar de água seja interrompido, na caixa ainda pode haver água para algum tempo de consumo, dependendo do histórico de consumo e fornecimento. E, se o fornecimento for restabelecido antes de que o consumo seja afetado, esse "entupimento" momentâneo pode passar desapercebido. Assim, a medida Pby não expressa nem a distribuição estática nem a dinâmica das ocorrências das operações dos tipos *wait*, *match* e *bypass*.

3.1.2 Paralelismo Médio

PM é uma medida "grosseira" da quantidade de paralelismo de um programa. Ela não mostra o comportamento do paralelismo durante execução do programa. Assim, PM alto não significa um paralelismo constante, pois pode haver concentrações de paralelismo numa parte do código e baixos níveis no resto. Neste caso, é impossível manter um número de UPs igual ao valor do PM ocupadas durante todo o tempo de execução.

O grafo do programa Programa-Recursivo encontra-se na Figura 3.8. Este programa é um dos programas duplamente recursivos mais simples e possui um alto valor de fichas de dados que passam direto pela UE. Seu Pby é igual a 82%. A silhueta da variação temporal do seu paralelismo, que corresponde ao caso quando se pode usar um número ilimitado de UPs a qualquer momento, é mostrada na Figura 3.9. Observa-se que existe uma concentração do paralelismo no meio da execução.

O Programa-Recursivo foi simulado para diferentes números de iterações, produzindo diferentes níveis de paralelismo. O gráfico da Figura 3.10 mostra as respectivas curvas de desempenho, cujo comportamento é quase o mesmo que o das curvas do gráfico da Figura 3.7. Será, então, que a perda de desempenho do gráfico da Figura 3.10 também é causada por concentrações de operações *wait/match*?

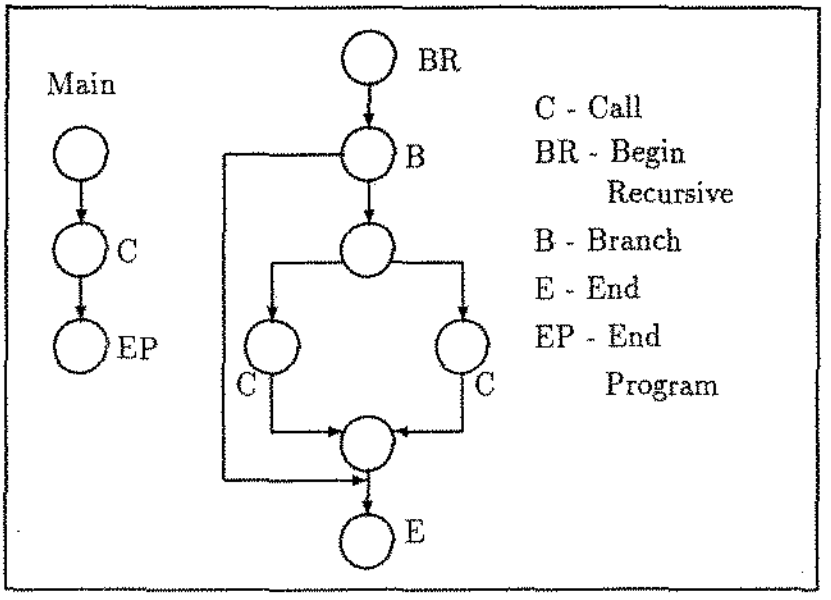

Figura 3.8: Grafo do Programa-Recursivo. Um dos programas duplamente recursivos mais simples. Pby igual a 82%.

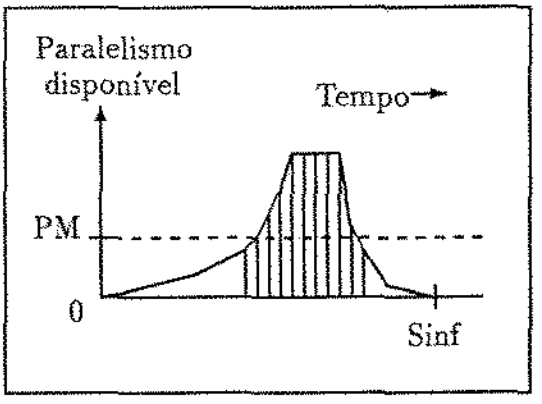

Figura 3.9: Variação temporal dos programas duplamente recursivos.

Para tirar essa dúvida, o Programa-Recursivo foi simulado para o mesmo conjunto de iterações anterior, no sistema sem "gargalos" no anel. Ou seja, da saída à entrada da UX, todos os pacotes são processados imediatamente. Nestas condições, a UX não pode ficar sem pacotes devido à lentidão das outras unidades.

O gráfico da Figura 3.11 mostra as curvas de desempenho do Programa-Recursivo na MFDM sem "gargalos" no anel. Estas curvas são iguais àquelas obtidas com a máquina padrão. Assim, nem a UE nem outra unidade podem ser a causa da aparente perda de desempenho. Conclui-se daí que a causa somente pode ser uma falta de paralelismo no programa.

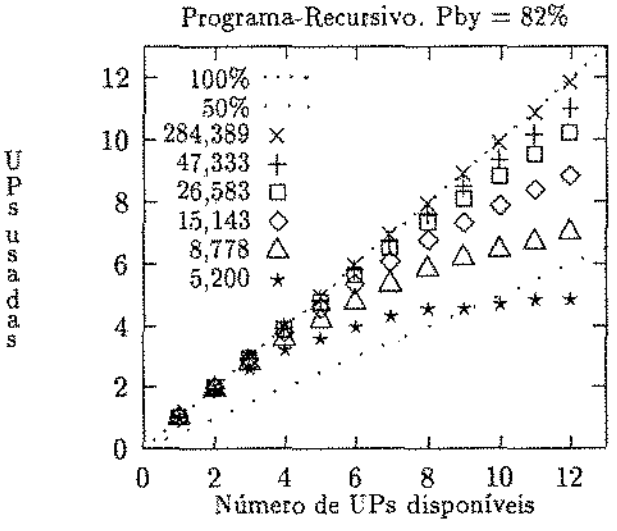

Figura 3.10: Curvas de desempenho do Programa-Recursivo com diferentes níveis de PM.

Como na MFDM somente estão disponíveis instruções com no máximo dois argumentos de saída e dois de entrada, tanto a expansão como a compressão do paralelismo são processos que requerem vários níveis no código. Então, a menos que o programa tenha paralelismos inicial e final altos, como os Programas-Constantes, sempre existe, pelo menos nestes trechos, baixo paralelismo. E, devido às dependências de dados, não é possível compensar o paralelismo nestes pontos. Exemplo disto são os programas da classe do Programa-Recursivo. Nestes casos, a perda de desempenho citada nas publicações não é causada por nenhum mau funcionamento da máquina. Simplesmente não é possível manter em cons-

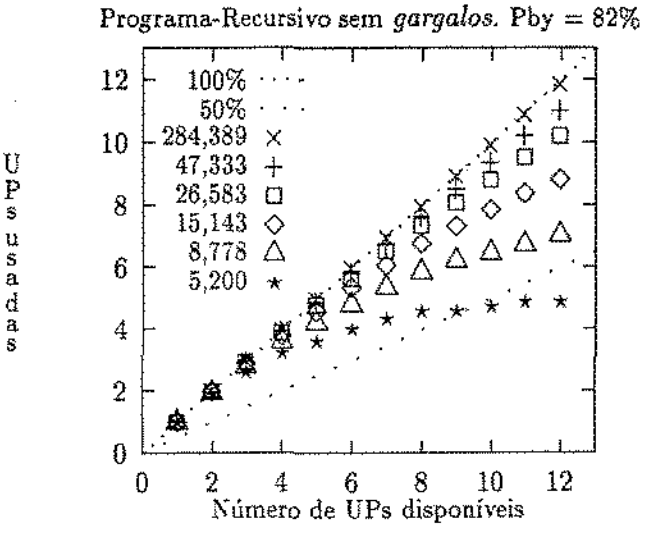

Figura 3.11: Curvas de desempenho do Programa-Recursivo com diferentes níveis de PM na MFDM sem "gargalos" no anel.

tante funcionamento um número alto e constante de UPs durante a execução deste tipo de código.

A medida PM fornece uma idéia do número ótimo de UPs que seriam necessárias para a execução de um programa, se este possuir uma distribuição constante do paralelismo no seu código. Contudo, é necessário conhecer a distribuição real do paralelismo do programa, para que se possa ter uma idéia mais realista do efeito da execução quando há um grande número de UPs disponíveis.

3.2 Unidade de Emparelhamento

A UE tem sido responsabilizada pela perda de desempenho nas publicações que analisam a máquina de Manchester, p. ex. [GW83, GKW85]. Cada pacote produzido pela UE, cuja memória geralmente apresenta altos níveis de ocupação, corresponde a uma instrução que fica disponível à UX. Assim, pode-se observar a UE como produtor e a UX como consumidor, mas com a característica de que, ao invés de uma fila, entre elas existem duas: uma entre a UE e a UAI, outra entre a UAI e a UX. Estas filas são o conjunto de *buffers* existentes entre estas unidades (ver 2.1).

Como se observa na Tabela 2.2, com até 12 UPs em uso, a UE em operações *bypass* e a UAI não são "gargalos" no anel, ambas possuindo capacidade de produção superior à possível demanda, limitada pelo número de UPs. Contudo, o mesmo não ocorre para as operações *wait* e *match* da UE. Quando ocorre uma delas, o tempo gasto é quase eqüivalente ao tempo gasto em cinco operações *bypass*. Só que, numa operação *match*, ocorre a produção de um pacote e a conseqüente liberação de uma instrução para execução, o que não ocorre numa operação *wait*. Na execução completa de um programa, essas três operações geralmente ocorrem misturadas. Por exemplo, toda operação *match*, é necessariamente precedida por uma *wait*. Contudo, pode haver seqüências com concentrações de duas ou mais destas operações.

Em seqüências de *bypass*, o consumo da UAI, devido às diferenças de *throughput* das unidades, passa a não acompanhar a produção de pacotes da UE. Entretanto, isto não afeta o sistema, pois a UAI tem *throughput* superior ao consumo da UX. Assim a ocupação das filas entre as UE e UAI, UAI e UX, passa a crescer. Em seqüências de *match*, o ritmo de produção de pacotes da UE é inferior ao de consumo da UAI e, assim o fornecimento à UX fica dependente do *throughput* da UE. Isto só passa a representar um problema, quando existem mais de 10 UPs disponíveis, quando a oferta de pacotes passa a ser inferior à demanda de instruções, como mostra a Tabela 2.2. Já em seqüências de *wait*, a produção de pacotes da UE é zero e conseqüentemente as filas entre a UE e a UX tendem a se esvaziar, comprometendo a oferta de instruções à UX. Assim, seqüências de operações *match* e, principalmente, *wait* devem ser evitadas. Isto significa intercalá-las com operações do tipo *bypass*.

As operações que envolvem a manipulação da memória da UE caracterizam-se por sua lentidão [Kir90]. Assim, a velocidade da UE determina, na maioria das vezes, o desempenho máximo que o anel pode atingir. A sobrecarga do dispositivo de armazenamento da UE não tem nenhum aspecto positivo. Além de requerer mais memória, comprometendo a eficiência dos tempos de manipulação dos dados, pode forçar o uso da UT. Acessos à UT tornam as operações *wait* e *match* ainda muito mais lentas e problemáticas.

Entretanto, a sobrecarga da memória da UE pode ser encarada como um sinal de outro problema. Altos índices de utilização, significam que várias operações *wait* foram realizadas precocemente e que as correspondentes operações *match* tenderão a ocorrer concentradas. Assim, o *throughput* da UE tende a se manter baixo, o que pode comprometer o uso das UPs disponíveis e, por conseguinte, a eficiência da máquina.

Na MFDM existe um desequilíbrio entre o *throughput* da UE, em operações *wait* e *match*, e o das outras unidades. Assim, p. ex., a UE é um problema nos

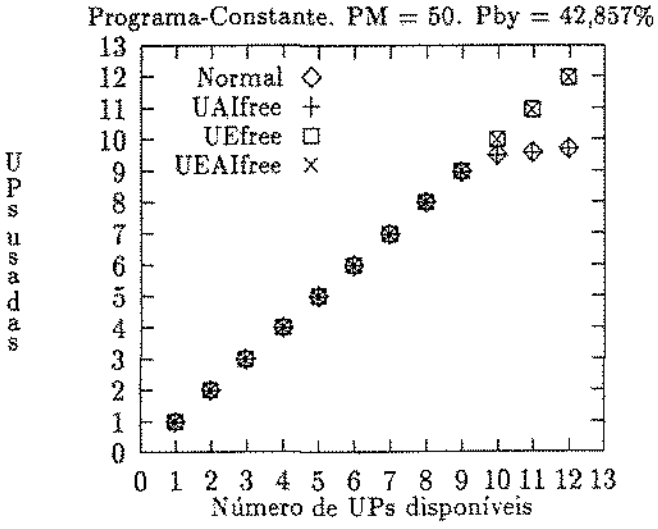

Figura 3.12: Curvas de desempenho do Programa-Constante do grupo 1 com Pby = 42,857%, para diferentes estados da MFDM.

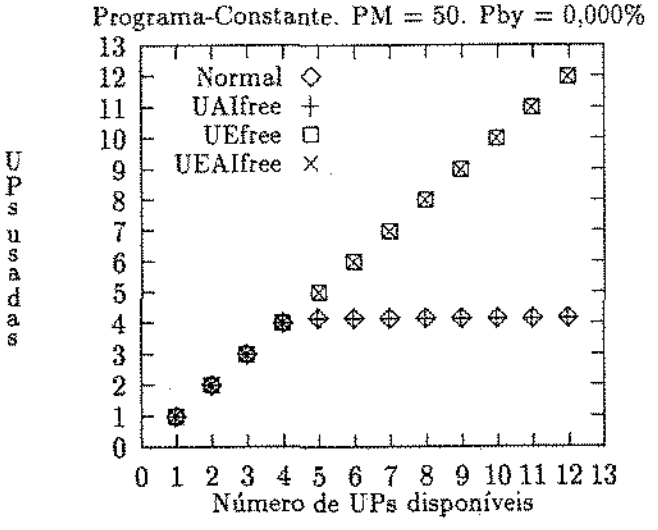

Figura 3.13: Curvas de desempenho do Program-Constante do grupo 1 com Pby = 0,000%, para diferentes estados da MFDM.

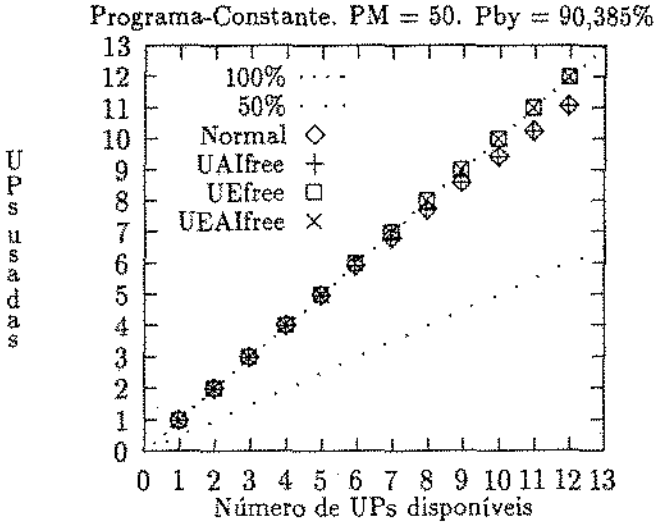

Figura 3.14: Curvas de desempenho do Programa-Constante do grupo 2 com Pby = 90,385%, para diferentes estados da MFDM.

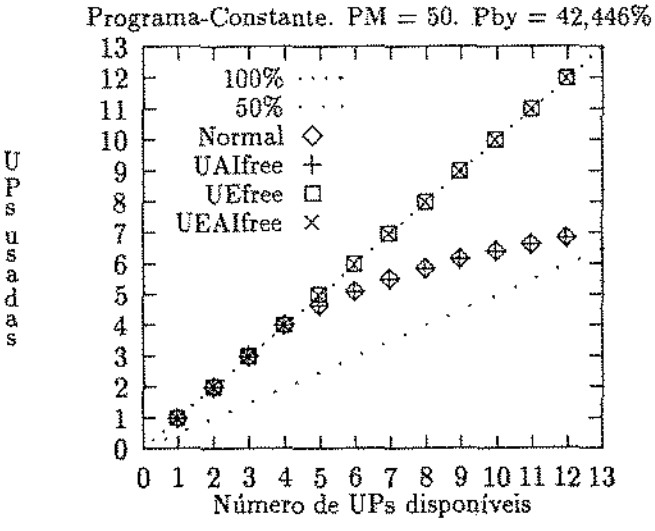

Figura 3.15: Curvas de desempenho do Programa-Constante do grupo 2 com Pby = 42,446%, para diferentes estados da MFDM.

casos descritos em 3.1.1. Para demonstrar que a UE é o único problema nessas execuções, os gráficos das Figuras 3.12, 3.13, 3.14 e 3.15 mostram as execuções dos Programas-Constantes para os exemplos críticos. Estes gráficos mostram as curvas de desempenho correspondentes a quatro diferentes estados da máquina: com a UAI com seus tempos dilatados (UAIfree), com a UE com seus tempos dilatados (UEfree), com ambas com tempos dilatados (UEAIfree) e com a máquina com seus tempos normais (Normal). Compreende-se por tempos dilatados que a unidade tem capacidade de consumir imediatamente todas as entradas que lhe forem fornecidas.

Observa-se nessas curvas que o comportamento da máquina é idêntico para os casos Normal e UAIfree e para os casos UEfree e UEAIfree. Nestes dois últimos, as curvas de desempenho não apresentam perdas, mostrando 100% de utilização da UPs disponíveis. Entretanto, o mesmo não ocorre nos outros dois casos, cabendo, pois, concluir que aqui a UE é o único "gargalo" do sistema.

3.3 Escalonamento FIFO

Na execução de um programa, não há como diminuir o número de operações da UE ou alterar os seus tipos. Ambos são característicos do grafo. As instruções de dois operandos vão causar operações *wait* e *match* invariavelmente, como as de um só operando causarão operações *bypass*. Existe um relacionamento e uma dependência entre as execuções na UX e as operações da UE, ambas condicionadas pela topologia do grafo. Assim, como a MFDM é regida pelos dados, é necessário gerenciar a ordem destes para que se possa obter desempenho adequado.

Nas unidades da MFDM, as fichas de dados são sempre tratadas pela ordem de chegada. Os resultados das instruções executadas são colocados nas filas entre a UX e a UE na ordem em que foram geradas. A ordem em que os pacotes requisitam os serviços da UE decorre da ordem em que foram gerados e não daquela em que serão consumidos. Assim, caminhos não-críticos do grafo podem ser percorridos precocemente. Além disso, esses percursos acabam por ficar bloqueados em instruções de dois operandos, um dos quais resulta de um caminho ainda não percorrido.

Essa situação decorre da técnica FIFO adotada, que procura habilitar o maior número possível de caminhos, num modelo de execução *breadth-first* [RS87]. Todas as fichas produzidas por vértices do mesmo co-nível[2] serão processadas pela UE antes das fichas produzidas pelos vértices do próximo co-nível no grafo do

[2] O co-nível (*co-level*) de um vértice é igual ao comprimento do maior caminho crítico entre ele e o vértice inicial.

programa. Então, existe uma tendência à produção de fichas cujos parceiros somente estarão disponíveis em ciclos posteriores. Por isto, as primeiras fichas têm que parar na UE e requisitar uma operação *wait*. Isto representa um uso desnecessário dos recursos de armazenamento da UE e a tendência de concentrações de operações *wait/match*.

Esse comportamento independe do número de UPs disponíveis, já que o acesso das fichas à UE continua regido pela sua produção, mas a ordem de execução das instruções é regido pela ordem das fichas. Por isto, observa-se uma sobrecarga da memória da UE quase insensível à variação do número de UPs disponíveis.

Com o objetivo de mostrar esses fatos e suas conseqüências, utilizou-se o programa SUM (ver Figura 3.16), baseado no problema da somatória duplamente recursiva[3]:

$$\sum_{i=0}^{j} a[i] = \sum_{i=0}^{z} a[i] + \sum_{g=z+1}^{j} a[g],$$

$$\text{com } 0 < z < j,$$

Usando-se a propriedade de recursão, o problema de somatória duplamente recursiva tem uma implementação simples. Uma possível versão desta é ilustrada a seguir:

```
SUM (a, i, j) {
    if (i = j)
        return a[i];
    else
        return SUM(a, i, ⌈(i+j)/2⌉ − 1) + SUM(a, ⌈(i+j)/2⌉, j);
}
```

Mesmo com o grafo de execução crescendo rapidamente por causa das recursões, SUM exibe trechos de baixo paralelismo no começo e no final da sua execução, como o Programa-Recursivo em 3.1.2. Esse trechos tornam impossível a utilização constante de todos as UPs disponíveis. Para evitar isto, usou-se um programa composto de vinte versões paralelas de SUM: SUM20. Assim, expande-se o paralelismo do programa, tornando-o capaz de manter pelo menos 20 UPs constantemente ocupadas.

O gráfico da Figura 3.17 mostra as curvas de desempenho de SUM20. Diferentes números de iterações foram usados para produzir diferentes níveis de paralelismo. A perda de desempenho observada não pode ser atribuída à falta

[3] *Double-recursive integer sum problem.*

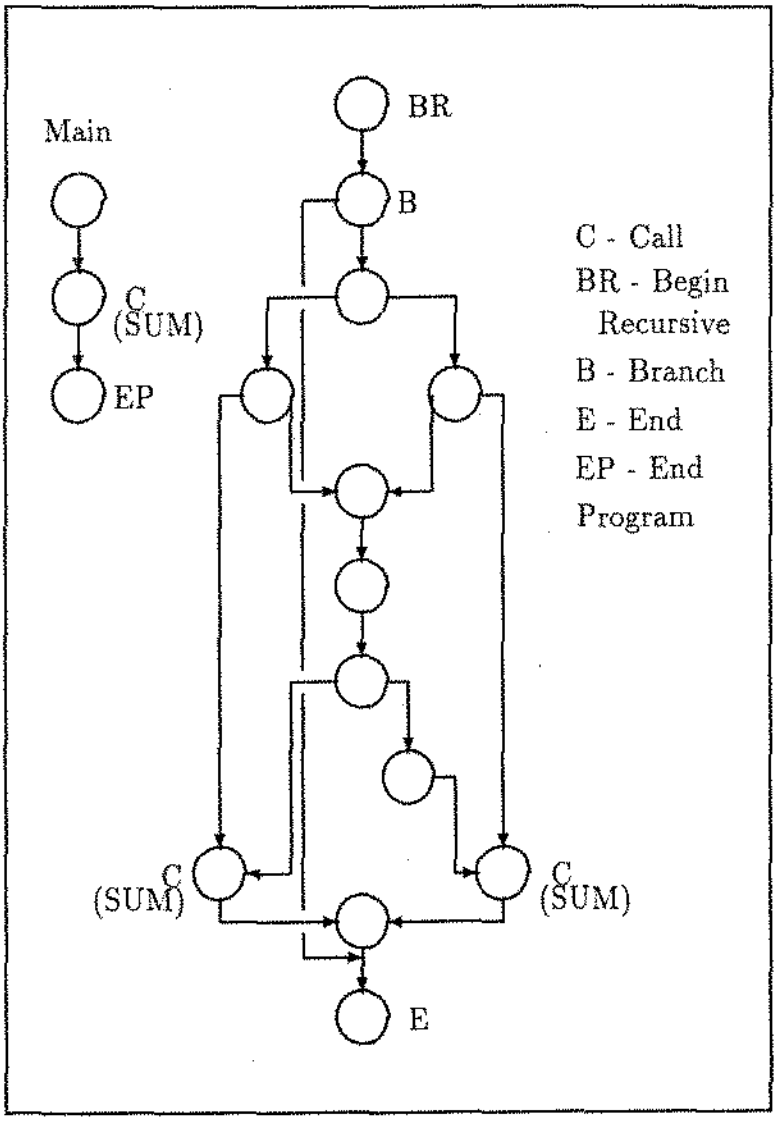

Figura 3.16: Grafo do programa SUM.

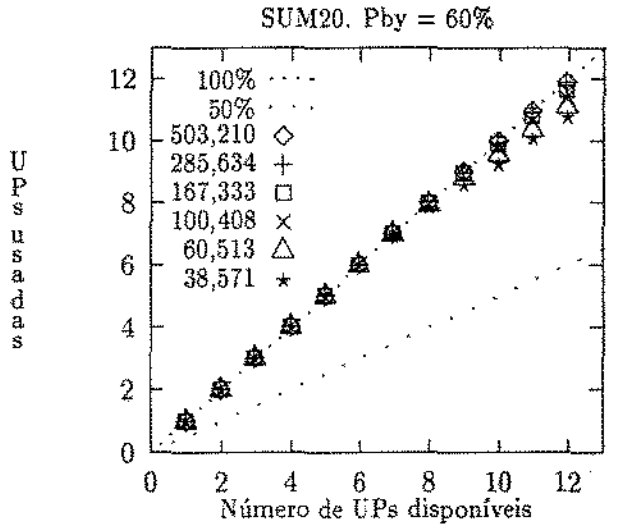

Figura 3.17: Curvas de desempenho do programa SUM20 com diferentes níveis de PM.

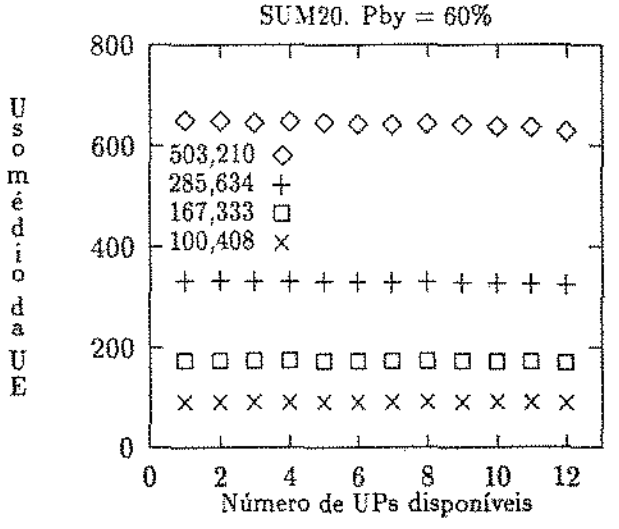

Figura 3.18: Curvas da utilização média da memória da UE pelo programa SUM20 com diferentes níveis de PM.

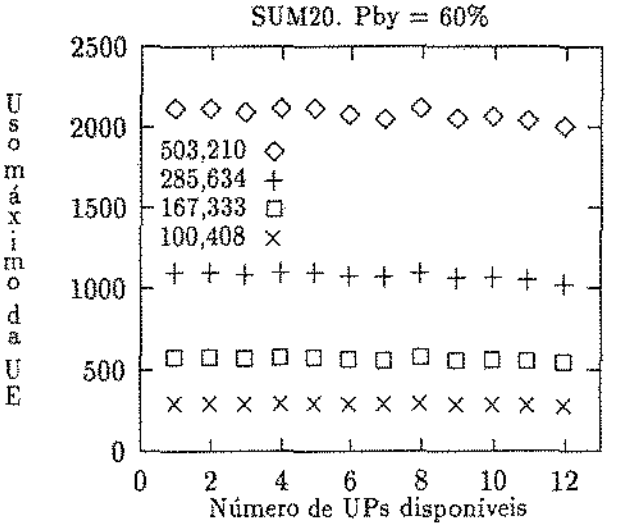

Figura 3.19: Curvas da utilização máxima da memória da UE para o programa SUM20 com diferentes níveis de PM.

de paralelismo, pois este é constantemente alto durante toda a execução. O gráfico da Figura 3.18 mostra as curvas de utilização média da UE, "uso médio da UE" *versus* "número de UPs disponíveis", das quatro execuções de maiores PM. E o gráfico da Figura 3.19 mostra as correspondentes curvas de utilização máxima da UE, "uso máximo da UE" *versus* "numero de UPs disponíveis". Todas as execuções exibem altos índices de utilização do dispositivo de memória da UE. Isto mostra que existem concentrações de operações *wait* e, mais tarde, de operações *match*. Como foi explicado anteriormente, essas concentrações reduzem o *throughput* da UE, o que pode deixar as UPs ociosas.

Outro fato observado é o efeito da ordem de execução *breadth-first* na execução dos programas. Independentemente do número de UPs disponíveis, a ordem dos pacotes no anel é mantida basicamente a mesma, o que mantém constante o nível de utilização da memória da UE.

O gráfico da Figura 3.20 mostra a curva de utilização da UE, "número de fichas em espera" *versus* "número de ciclos da UE", com 12 UPs disponíveis de SUM20 com PM igual a 38,571. E o gráfico da Figura 3.21 mostra a curva de utilização das UPs correspondente, "UPs usadas" *versus* "número de ciclos de execução". Mesmo com paralelismo mínimo igual a 20, existem ciclos nos quais

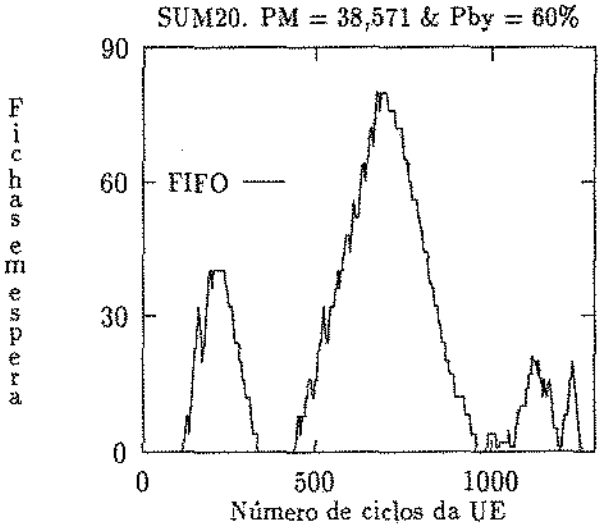

Figura 3.20: Curva de utilização da UE na execução de SUM20 com PM = 38,571 é 12 UPs disponíveis.

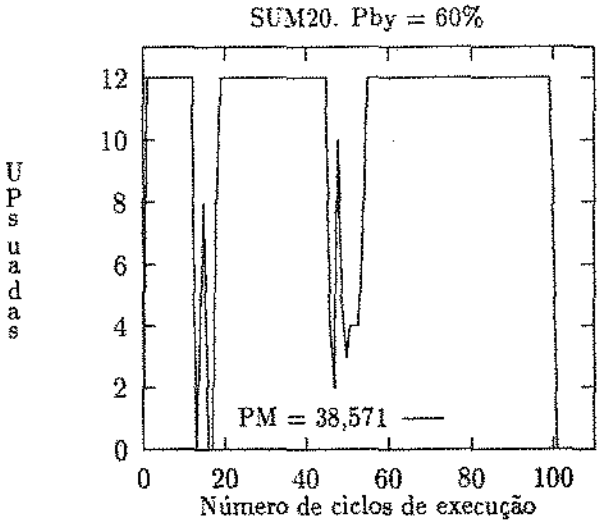

Figura 3.21: Curva de utilização das UPs na execução de SUM20 com PM = 38,571 e 12 UPs disponíveis.

até mesmo todas as UPs estão ociosas. Este fato somente pode ser provocado por uma perda na taxa de fornecimento de pacotes executáveis, causada pela oscilação dos níveis de vazão da UE. O rápido aumento do número de fichas que aguardam par observado na Figura 3.20 corresponde a concentrações de operações *wait* e os abruptos declínios, a concentrações de operações *match*. Os efeitos dessas concentrações na UX são visíveis no gráfico da Figura 3.21.

A técnica de escalonamento FIFO originalmente adotada na MFDM produz uma ordenação de pacotes que geralmente sobrecarrega a memória da UE e pode gerar concentrações de operações problemáticas, o que pode tornar a UE o "gargalo" do sistema.

3.4 Definição do Problema

Com base nos testes já realizados, observa-se que os parâmetros PM e Pby não são métricas absolutas. Assim, a análise e a classificação de programas apenas por intermédio delas, constitui um erro. Também identifica-se a UE como um problema para o sistema somente em certas situações, não sendo ela a eterna responsável pela perda de desempenho descrita nas publicações. Além disso, mostra-se que a técnica de escalonamento FIFO original da máquina constitui outro problema para o sistema.

As análises de comportamento e eficiência da execução de um programa não podem ser baseadas somente nos valores de PM e Pby, como também não é possível deduzir previamente o comportamento dinâmico de um programa unicamente através do conhecimento desses parâmetros. Eles não dão uma noção adequada da distribuição do paralelismo e das operações *bypass* no programa e durante sua execução. Assim, programas com o mesmo valor de PM ou Pby não têm necessariamente o mesmo comportamento. A perda de desempenho na execução de diferentes programas, mesmo quando esses parâmetros possuem valores eqüivalentes, deve ser provocada por outros motivos.

Para uma configuração com até 12 UPs disponíveis, a MFDM tem como ponto fraco a UE, que é sensível à solicitação concentrada de operações *wait* e *match*. E, neste sentido, a técnica de escalonamento FIFO constitui-se num dos geradores dessas concentrações. Contudo, a especialização do escalonamento não pode evitar concentrações causadas pelo código do programa e protegidas pelas dependências de dados.

Alguns programas, no entanto, não têm problemas com a UE. O Programa-Recursivo é um exemplo para cuja execução a MFDM é absolutamente competente, mesmo não sendo ele indicado para esse tipo de máquina. Contudo,

excluindo-se inapropriados, existem outros cujo código exibe configurações que provocam a sobrecarga da UE. A MFDM não é competente para a execução desses programas, entre os quais se incluem os Programas-Constantes do grupo 2. Apenas a alteração dos códigos, quando possível, ou a alteração das características do *hardware* da máquina poderiam contornar esta incapacidade.

O restante dos programas tem sua execução prejudicada pela técnica de escalonamento FIFO adotada pela MFDM. Ela sobrecarrega a UE e, por conseguinte, provoca uma perda de desempenho. Neste sentido, pode-se resolver este problema sem recorrer à inclusão de novas unidades, que aumentariam o *pipe* e a possibilidade do aparecimento de "bolhas" [Nav90, Das89]. A alteração da política de escalonamento, para otimizar o uso das unidades da máquina pode resolver satisfatoriamente este problema interno da MFDM.

Este estudo caminha nessa direção. No próximo capítulo, estuda-se o problema de escalonamento de tarefas dando-se ênfase aos modelos baseados em listas. E no capítulo 5, são analisados os impactos da mudança do escalonamento na MFDM.

Capítulo 4

Escalonamento de Tarefas

"Não queime pontes.
Você ficará surpreso ao descobrir
quantas vezes tem de atravessar o mesmo rio."

"De vez em quando
pegue o caminho mais bonito."

Qualquer problema pode ser subdividido em subproblemas, que também podem ser subdivididos, num processo recursivo, até se atingir unidades de solução elementar. Inversamente, estas unidades combinam-se ordenadamente para compor o problema inicial.

Considerando que exista apenas um indivíduo para executar a solução de todo um conjunto de unidades, é necessário seqüencializar as unidades, respeitando suas dependências. Caso exista mais de um indivíduo, várias unidades podem ser alocadas a executores distintos e, aí, executadas simultaneamente, sempre respeitando suas dependências. O processo que corresponde a definição da ordem de execução das unidades e escolha dos executores aos quais elas serão alocadas é denominado escalonamento [Wil93].

A maioria dos escalonadores busca obter o máximo de paralelismo possível e manter as UPs sempre ocupadas. Com este procedimento, pretende-se conseguir menores tempos de execução. Na prática, entretanto, outros fatores demonstram merecer destaque, como, por exemplo, os atrasos de comunicação entre as UPs.

Uma propriedade importante num algoritmo de escalonamento eficiente é que o tempo total de execução de um programa não deve aumentar se houver um

aumento do número de UPs disponíveis. Se a adição de UPs ao sistema provocar a degradação do desempenho, o escalonador deve ser capaz de optar por não usar as UPs adicionais [KL92a].

Este capítulo resume uma visão geral do problema de escalonamento de tarefas. Na seção seguinte, o problema é descrito mais formalmente e, em seguida alguns de seus aspectos são abordados brevemente. A importância da consideração dos custos de comunicação também receberá atenção especial. Por fim, são resumidas algumas técnicas de escalonamento baseadas em listas. E, a técnica CP/MISF/∞, desenvolvida durante este trabalho para a MFDM, durante este estudo, é descrita detalhadamente.

4.1 O Problema de Escalonamento de Tarefas

O problema do escalonamento de tarefas em um sistema paralelo pode ser definido da seguinte maneira:

- Sejam T um conjunto de n tarefas $\{t_1, t_2, \ldots, t_n\}$ e $\prec$ uma relação de precedência que estabelece uma ordem parcial entre as tarefas. Se $t_i \prec t_j$ (t_i precede t_j), t_j só pode ser executada após a execução de t_i. Cada tarefa t_i tem um tempo de execução $te(t_i)$ associado. Para todo par de tarefas t_i e t_j, onde t_j é a sucessora imediata de t_i, existe um conjunto M_{ij} de $|M_{ij}|$ mensagens de t_i para t_j $\{M_{ij}[1], M_{ij}[2], \ldots, M_{ij}[|M_{ij}|]\}$. Estes componentes podem ser representados num grafo orientado (ver Figura 4.1), onde:

  - T corresponde ao conjunto de vértices $V = \{v_1, v_2, \ldots, v_n\}$.
  - $\prec$ define a existência das tarefas, sendo que, se $t_i \prec t_j$, existe uma aresta orientada de v_i para v_j.
  - A cada vértice v_i está associado o valor $te(t_i)$.
  - A cada aresta (v_i, v_j) está associado o conjunto M_{ij}, correspondente ao conjunto de mensagens entre as tarefas t_i e t_j.

- Um conjunto P de p unidades de processamento idênticas $(up_1, up_2, \ldots, up_p)$. O tempo requerido para a transmissão de uma mensagem da unidade de processamento up_s para a up_r é representado por $tcom(up_s, up_r)$. Assim sendo, existe um atraso de comunicação $(|M_{ij}| * tcom(up_s, up_r))$ entre a tarefa t_i e sua sucessora imediata t_j, caso t_i seja alocada a up_s e t_j a up_r.

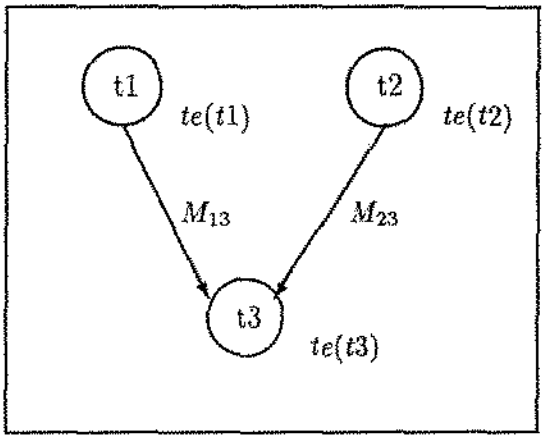

Figura 4.1: Grafo orientado representando as características do problema de escalonamento de tarefas.

- Uma ou mais funções-objetivo, como, p. ex., minimizar o tempo de execução[1] de T ou maximizar o uso das unidades de processamento.

O objetivo do escalonamento é a execução das n tarefas nas p unidades de processamento, satisfazendo a relação de precedência $\prec$ e atendendo às funções-objetivo [Gar92].

4.2 Aspectos de Escalonamento

Vários aspectos podem ser analisados no estudo e na definição do escalonamento de tarefas. O fato de se considerar todas as UPs idênticas é um exemplo. A descrição e análise sucinta de alguns destes aspectos são realizadas a seguir.

4.2.1 Estático ou Dinâmico

O escalonamento dos módulos é realizado com base na partição do programa e nos parâmetros da máquina alvo. Ele é dito estático quando ocorre em tempo de compilação e dinâmico quando ocorre em tempo de execução.

O escalonamento estático é livre de custos em tempo de execução, mas é inadequado ao tratamento de laços e desvios [ERL90]. Já o escalonamento dinâmico

[1] *Makespan.*

não aproveita técnicas mais elaboradas que necessitam de grande processamento ou retrocessos para escolha de uma ordem vantajosa para execução dos módulos. A união destes estilos proporciona um tratamento adequado para laços e desvios [BG90a, Tow86] e o uso de métodos mais eficientes.

4.2.2 Determinístico ou Probabilístico

Para realizar o escalonamento estático, é necessário conhecer os valores dos tempos de execução das tarefas ($te(t_i)$), das necessidades de comunicação entre tarefas M_{ij} e dos custos de comunicação entre as UPs $tcom(up_s, up_r)$. Se estes valores são conhecidos com exatidão pode-se fazer um escalonamento determinístico. Se eles forem estimados com base em suposições probabilísticas, faz-se um escalonamento probabilístico. Em ambos os casos, estas informações necessitam estar disponíveis antes da execução, o que possibilita uma melhor escolha da técnica de escalonamento e um ajuste de seus parâmetros.

4.2.3 Ótimo ou Sub-Ótimo

Na grande maioria dos casos, o problema de escalonamento não admite solução ótima. Uma vez que o grande número de parâmetros e suas possíveis combinações o torna intratável [Ull75, Cof76]. Ainda assim, soluções ótimas podem ser obtidas em casos específicos, onde se considera uma série de restrições ao problema [Hu61, MC69].

Na inviabilidade da obtenção de soluções ótimas, grandes esforços têm sido voltados ao aperfeiçoamento de soluções sub-ótimas. Pode-se destas destacar duas categorias [CK88]: aproximadas e heurísticas.

Aproximado ou Heurístico

Técnicas de escalonamento aproximado necessitam das mesmas informações que as de escalonamento ótimo. Contudo, com base em critérios de definição de bons resultados, elas buscam soluções boas mas não necessariamente ótimas [CK88].

As técnicas heurísticas não necessitam de todos esses parâmetros. Elas compensam a escassez de informações com o conhecimento de certas características da arquitetura e do conjunto de tarefas. Observa-se um maior esforço de pesquisa nesta categoria, pelo fato de, na prática, não se dispor de todas as informações antes da execução do programa.

4.2.4 Preemptivo ou Não-Preemptivo

Ao ser iniciada a execução de uma tarefa, a possibilidade de sua interrupção e posterior retomada é o que se denomina preempção. Neste caso, devem ser levados em conta os custos dos mecanismos especiais para a troca do contexto de execução e sua posterior restauração, bem como os custos associados ao tempo gasto nesse processo. Para que a preempção seja benéfica, é necessário haver um equilíbrio entre o número de preempções e os seus custos operacionais. O escalonamento não-preemptivo é também chamado escalonamento básico[2].

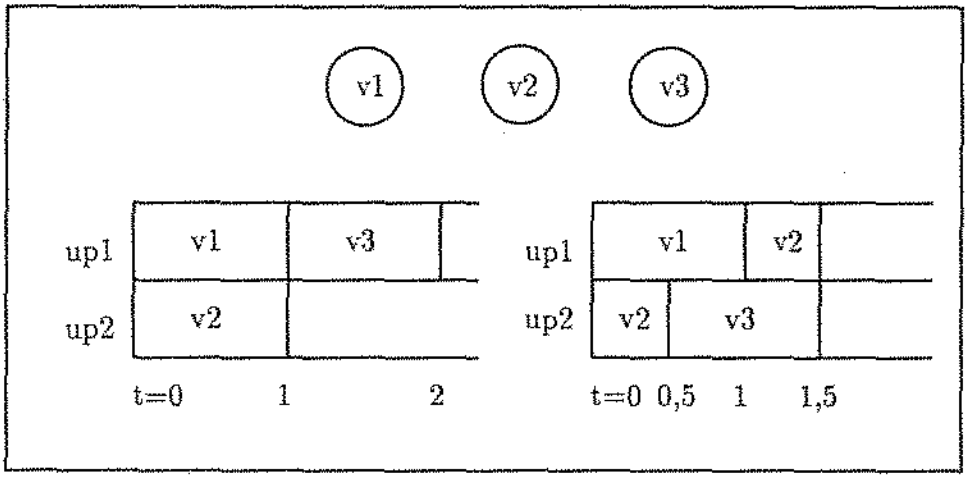

Figura 4.2: Ilustração da vantagem do escalonamento preemptivo. Todos os vértices possuem tempo de execução unitário.

A preempção baseia-se no fato de que com ela é possível a execução de tarefas em menor tempo do que no escalonamento básico, como exemplifica a Figura 4.2. Supõe-se nesse exemplo que não existam custos na preempção e nem atrasos de comunicação.

4.3 Custos de Comunicação

Na busca de algoritmos capazes de gerar um escalonamento mais próximo do ótimo, é indispensável a análise dos custos, ou atrasos, de comunicação do sis-

[2] *Basic schedule.*

tema. A desconsideração dos atrasos decorrentes do envio dos resultados de uma tarefa para outras dependentes, mas executadas em UPs distintas, pode gerar comportamentos inesperados na execução de programas paralelos.

Devido ao grande número de parâmetros a serem considerados no escalonamento, os custos de comunicação entre as UPs não tiveram, há até pouco tempo, destaque nas pesquisas da área. Maximizar a paralelização era a grande esperança para se obter menores tempos de execução. Entretanto, observa-se que a consideração daqueles custos pode trazer resultados positivos. Num sistema com grandes atrasos na comunicação entre as UPs, p. Ex., tarefas dependentes podem vir a ser mais eficientemente executadas se alocadas a uma mesma UP [KL88, LC92].

Na prática, o que se observa é que nem sempre o escalonamento com maior paralelismo é o mais eficiente. Distribuir módulos paralelizáveis para tantas UPs quantas possível tende a aumentar os atrasos de comunicação, o que contribui para alongar a execução [KL92a]. A Figura 4.3(a) reproduz o GFD da Figura 1.3(a), mas agora supondo que o custo da comunicação entre vértices alocados a uma mesma UP seja zero e entre vértices alocados a UPs distintas duas vezes o tempo de execução dos vértices. O escalonamento do *Gantt chart* da Figura 4.3(b) busca conseguir alto desempenho através da máxima paralelização. Devido aos atrasos de comunicação, em alguns casos é mais eficiente serializar a execução, como mostra o *Gantt chart* da Figura 4.3(c).

Com a finalidade de analisar o efeito dos atrasos de comunicação (AC), supõe-se nos exemplos a seguir um ambiente no qual:

- Qualquer vértice tem tempo de execução igual a TE.
- Nas comunicações internas a uma UP, AC é igual a zero. Esta consideração é comum nos trabalhos da área. Em geral, o custo das comunicações internas a uma UP são muito menores do que os existentes entre diferentes UPs.
- Nas comunicações entre quaisquer UPs distintas, os atrasos são iguais a AC.

Devido ao desnecessário *overhead* de comunicação, a tentativa de paralelizar os vértices de trechos seqüenciais somente aumenta o tempo de execução. Empacotar estes vértices num único módulo e executá-lo numa única UP, reduz sensivelmente os custos de comunicação. A Figura 4.4 exemplifica estas duas situações. A Figura 4.4(a) mostra a execução de um trecho seqüencial com os vértices alocados a UPs distintas e, portanto, sujeita a atrasos de comunicação. A Figura 4.4(b) mostra a mesma execução numa mesma UP e, portanto, livre

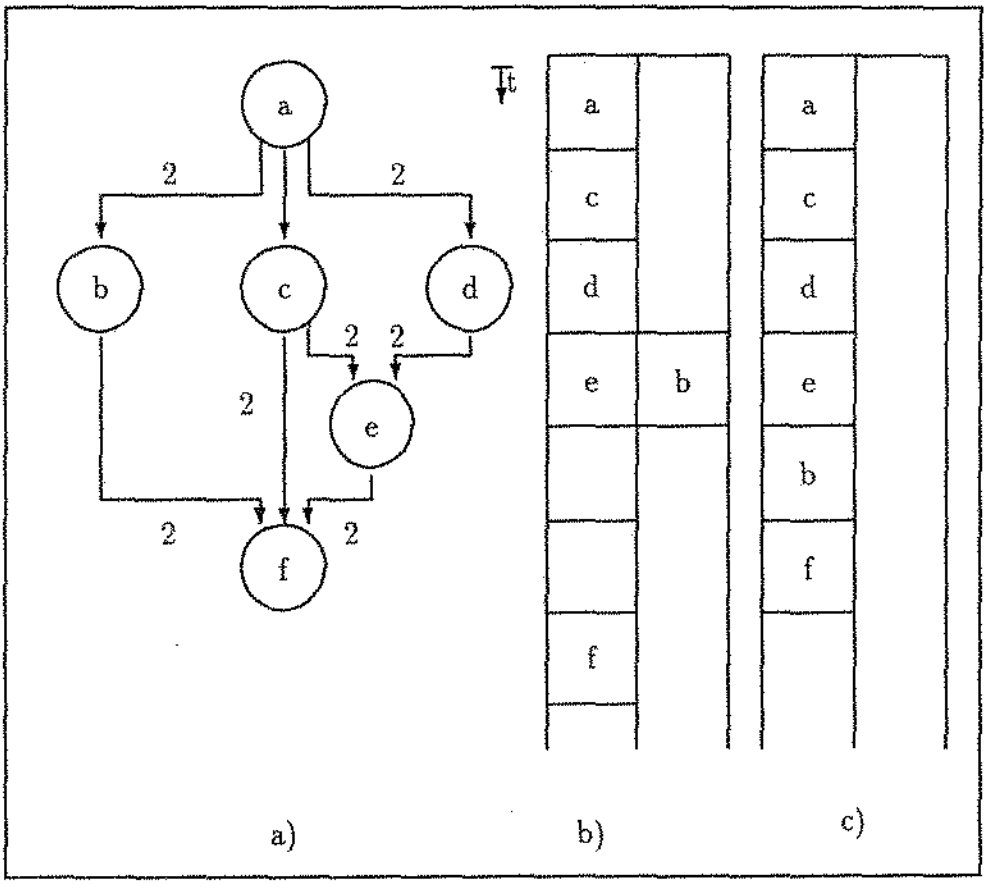

Figura 4.3: Impacto de uma política de escalonamento com análise dos atrasos de comunicação: a) GFD de um trecho de programa numa arquitetura com atrasos de comunicação iguais a duas unidades de tempo, b) *Gantt chart* de um escalonamento que enfatiza a paralelização e c) *Gantt chart* de uma política de escalonamento com análise dos custos de comunicação.

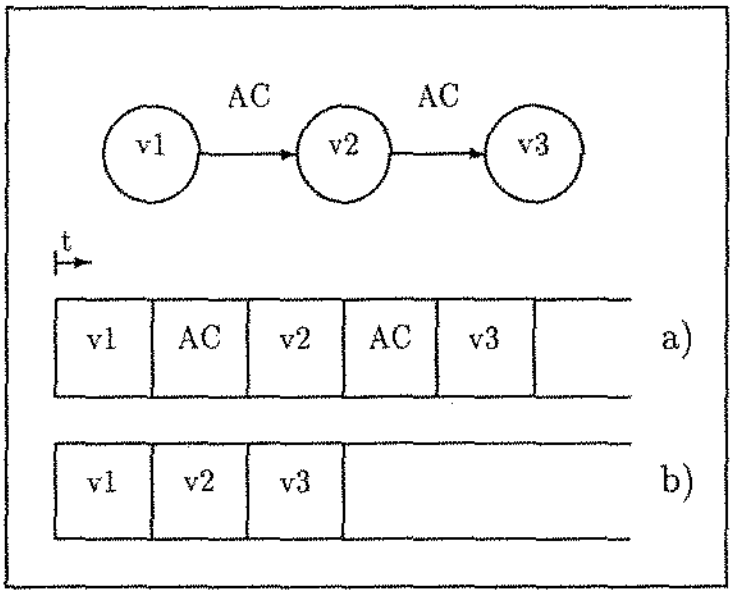

Figura 4.4: O efeito da tentativa de paralelização de um trecho seqüencial de programa.

dos atrasos. Como o trecho é seqüencial, tentar paralelizá-lo é pior que executá-lo seqüencialmente, devido aos atrasos de comunicação [VC92, Vis, YSK91]).

Em alguns casos, até mesmo o empacotamento de trechos paralelos pode melhorar o desempenho do sistema. A Figura 4.5 mostra o GFD de um código paralelo e o *Gantt chart* de um escalonamento paralelo. Variando-se a relação entre os atrasos de comunicação e o tempo de execução dos vértices, pode-se fazer uma análise dos casos onde a paralelização é melhor opção do que a serialização. Supondo-se as relações a seguir:

1. $TE < 2AC$ (ver Figura 4.6(a)).
2. $TE = 2AC$ (ver Figura 4.6(b)).
3. $TE > 2AC$ (ver Figura 4.6(c)).

Observa-se que para o caso 1, a execução seqüencial (ver Figura 4.6(d)) diminui o tempo de execução. Para o caso 2, ambas maneiras são eqüivalentes. E, no caso 3, a execução paralela, mesmo com os atrasos de comunicação, mostra-se mais eficiente.

Análise similar é possível considerando o empacotamento dos vértices v1 e v2 e de v3 e v4. A Figura 4.7 mostra os *Gantt charts* para as seguintes relações entre TE e AC:

1. $TE < AC$ (ver Figura 4.7(a)).
2. $TE = AC$ (ver Figura 4.7(b)).
3. $TE > AC$ (ver Figura 4.7(c)).

No caso 1, a execução paralela não é vantajosa, uma vez que é mais demorada do que a execução seqüencial. No caso 2, a execução em paralelo é eqüivalente à seqüencial, exceto por usar mais recursos. E, no caso 3, a execução em paralelo é mais rápida do que a seqüencial.

Algumas vezes, a execução de mais instruções pode diminuir o tempo de execução total. Baseada nesta idéia, a heurística *Duplication Scheduling Heuristic* (DSH) [KL88] (ver 4.4.12) usa o conceito de duplicação de tarefas para melhorar o desempenho de um sistema (ver Figura 4.8). A duplicação pode ter um impacto dramático no desempenho: duplicando-se os vértices v_1 e v_3 para que eles possam ser executados em ambas UPs evita qualquer atraso de comunicação (ver Figura 4.8(b)).

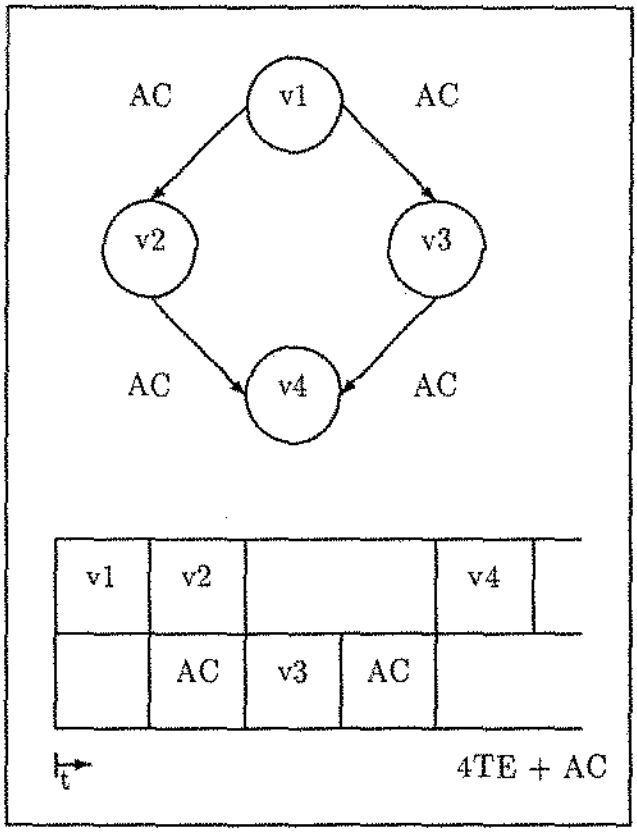

Figura 4.5: Exemplo de um trecho paralelo de programa e o *Gantt chart* de seu escalonamento paralelo.

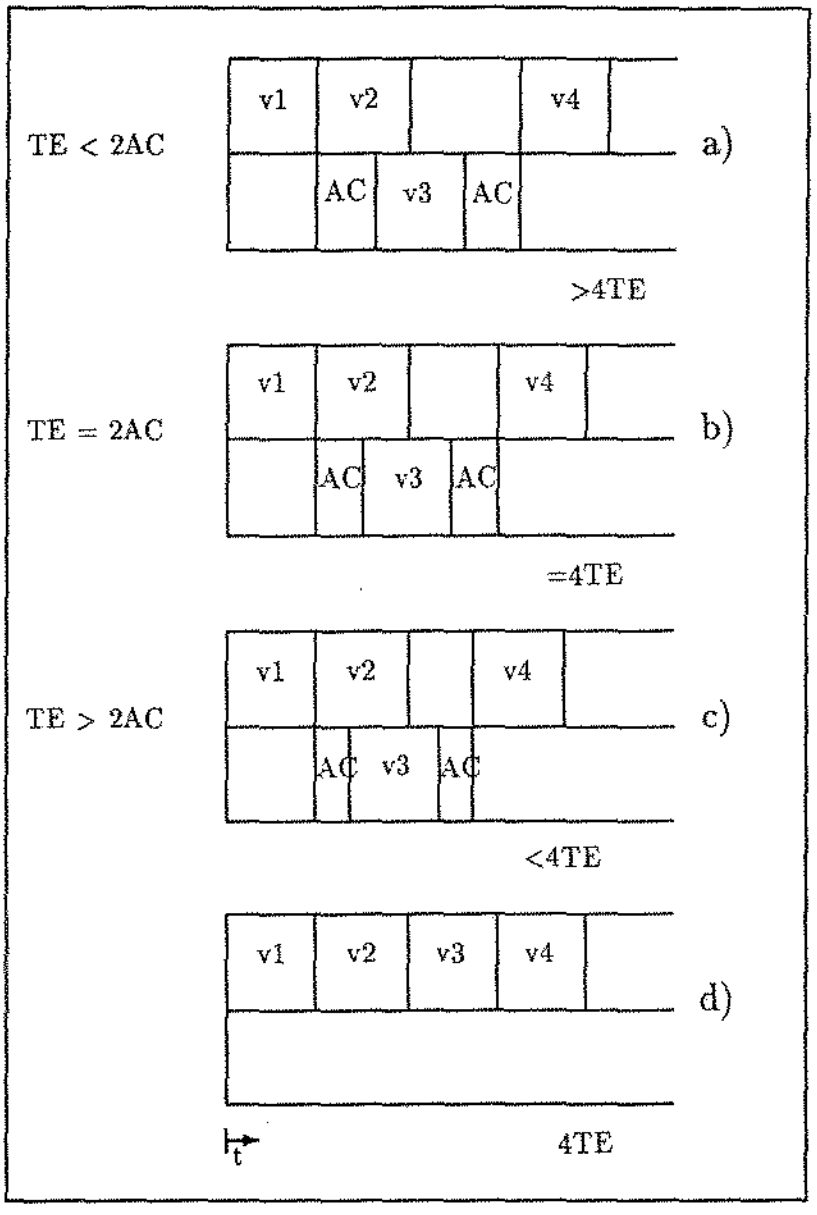

Figura 4.6: Análise do efeito dos Atrasos de Comunicação (AC) no escalonamento do trecho paralelo de programa da Figura 4.5.

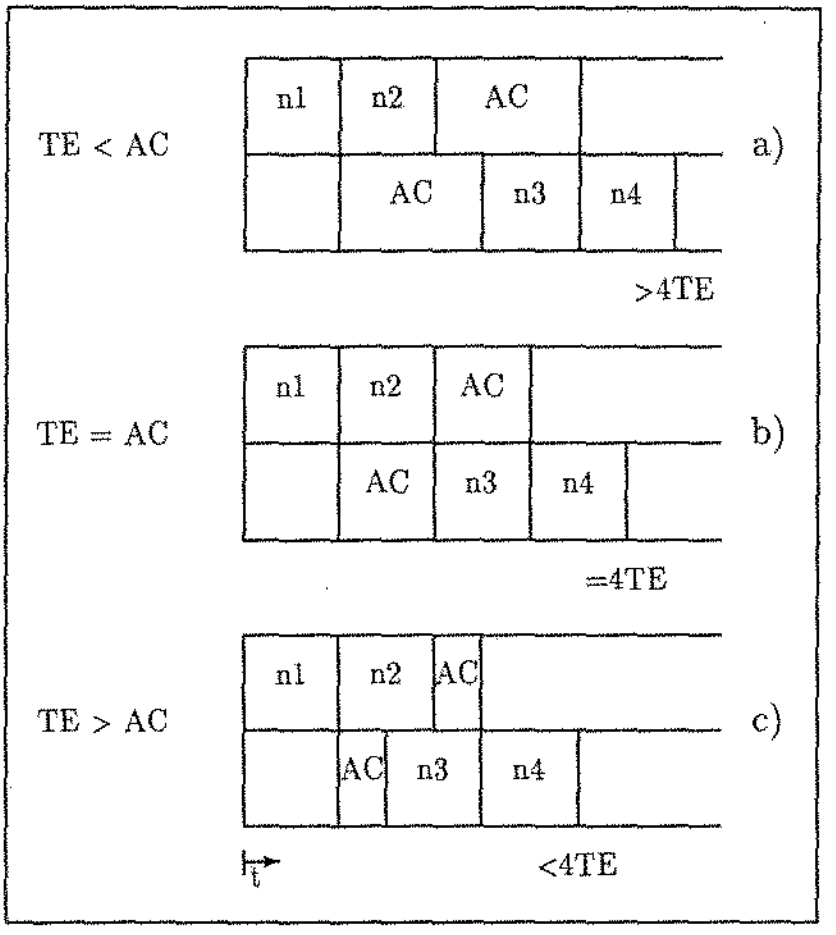

Figura 4.7: Análise do efeito do empacotamento de vértices no trecho paralelo de programa da Figura 4.5, considerando-se os Atrasos de Comunicação (AC).

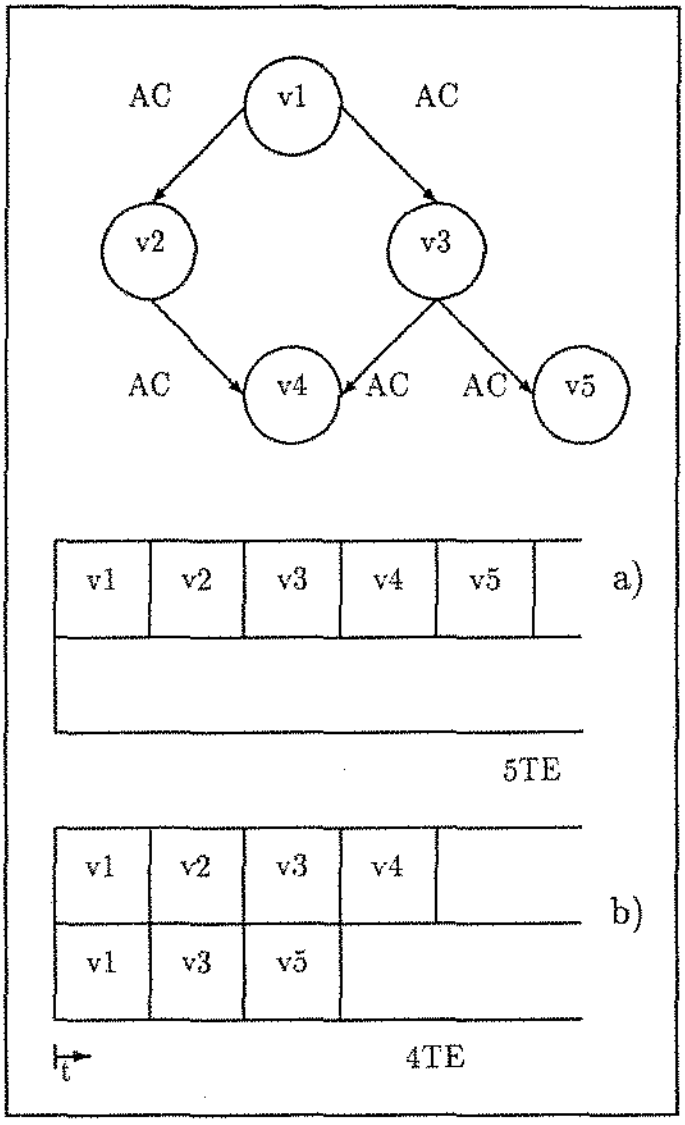

Figura 4.8: A duplicação pode ter um impacto dramático no desempenho: duplicando-se os vértices v_1 e v_3 para que eles possam ser executados em ambas UPs evita atraso de comunicação.

Algumas pesquisas recentes têm abordado políticas de escalonamento que consideram atrasos de comunicação [KL88, ERL90, LHCA88]. Na MFDM, a comunicação entre uma instrução e seus descendentes ocorre através da circulação dos pacotes de dados pelo anel. De modo geral, o tempo de circulação dos pacotes é cerca de 10 vezes maior do que o tempo de execução de uma instrução simples. Assim, pois, pode ser muito interessante evitar a comunicação através do anel [LC92], privilegiando-se comunicações internas à UX [VC92].

Existem muitas outras técnicas para aprimorar a partição dos programas e o empacotamento das instruções. Em [Vis], podem ser encontrados outros exemplos do impacto do empacotamento de instruções no desempenho do sistema.

4.4 Técnicas de Escalonamento

O escopo de aplicação das técnicas de escalonamento é muito grande e variado. Igualmente vasto é o número de trabalhos desenvolvidos. Os artigos [CS90, BFHS88, Gon77, CK88, Wil93] tratam da classificação e nomenclatura dos esforços neste campo.

A relação das variações das técnicas, políticas, funções-objetivo e máquinas que envolvem o problema de escalonamento de tarefas é extensa. As referências [YSK91, Sto77, Cof75, Her75] são alguns exemplos de trabalhos envolvendo diferentes arquiteturas. E [LY89, LHCA88, HCAL89, Kim87, DM80, Koh75] são alguns exemplos de trabalhos sobre diferentes funções-objetivo, p. ex.: minimizar os custos de comunicação, minimizar o tempo de execução, habilitar preempção.

Dentre as várias técnicas, a mais importante, por motivos históricos e por ser base para várias outras, é a lista de escalonamento[3]. Com base numa política de escalonamento[4] decorrente das funções-objetivo [PI77], as tarefas são rotuladas, o que possibilita uma análise de prioridades.

Quando todos os argumentos de uma tarefa estão disponíveis, esta é colocada numa lista de execução, de acordo com sua prioridade. Num processo "faminto" [5], assim que uma UP é liberada, a primeira tarefa da lista é alocada a ela. No caso de ser possível a preempção, verifica-se a existência de alguma tarefa em execução cuja prioridade seja menor do que as da lista. Existindo, ocorre a preempção dessa tarefa.

A seguir, são brevemente analisadas algumas técnicas de escalonamento. Todas são variações da política de prioridades da técnica de lista de escalonamento

[3] *Scheduling list.*
[4] *Scheduling rule.*
[5] *Greedy.*

ou técnicas baseadas nesta.

Não são aqui analisados os limites de execução[6] das técnicas de escalonamento. Para informações neste sentido, aconselha-se a análise dos artigos de cada trabalho descrito a seguir e os artigos [LS75, FL75, CR75, FB73, GG73, RCG72], que desenvolvem pesquisas mais específicas sobre o assunto.

4.4.1 *High Level First*

A técnica HLF [Hu61] consiste numa lista de escalonamento com a política de executar os vértices de maior nível[7] primeiro. O escalonamento obtido é ótimo, com o uso mínimo de UPs. Para isso, no entanto, há uma série de restrições:

- Todas tarefas têm tempo de execução iguais. Ou seja, os tempos de execução são de uma unidade de tempo ($(\forall i, te(t_i) = k) \Rightarrow k = 1$ unidade de tempo).
- Os grafos são do tipo *in-tree* [Kim87], árvore na qual todos os vértices possuem no máximo um sucessor imediato (ver Figura 4.9).

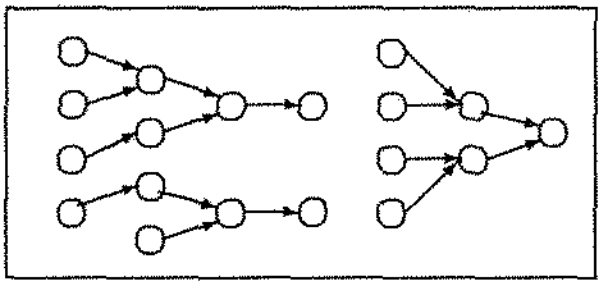

Figura 4.9: Relação de precedência do tipo In-Tree.

- Os atrasos de comunicação são desconhecidos e não considerados.
- Não há preempção.

O algoritmo é descrito a seguir:

[6] *Execution bounds.*

[7] O nível de um vértice v_i corresponde ao tamanho do maior caminho entre ele e o vértice terminal.

1. Rotular todos os vértices com $\alpha_i = x_i + 1$, onde x_i é o nível de v_i. Isto é facilmente realizado, iniciando-se o processo no vértice terminal (x_1, α_1), até os vértices iniciais, sempre prevalecendo o rótulo de maior valor. Observa-se que os rótulos crescem unitariamente, porque as tarefas têm tempo de execução unitário.

2. Se o número de vértices iniciais for menor ou igual a p, número de UPs, então todas as tarefas são alocadas. Caso contrário, as p tarefas iniciais de maior rótulo são alocadas. Em caso de conflito de duas tarefas de mesmo rótulo, e portanto mesma prioridade, a seleção é aleatória.

3. A etapa 2 é repetida no grafo restante, constituído dos vértices não alocados, até que ele se torne vazio.

A Figura 4.10 ([Hu61]) exemplifica esta técnica para uma arquitetura com três UPs.

4.4.2 *Highest Level First with Estimated Times*

A técnica HLFET ([ACD74]) parte da premissa de que os tempos de execução das tarefas são conhecidos ou estimados. Assim, as tarefas são rotuladas de uma forma semelhante à que ocorre na técnica HLF, só que $\alpha_i = x_i + te(t_i)$, onde x_i é o maior valor entre os rótulos dos filhos do vértice v_i.

HLFET obtém soluções sub-ótimas, não considera os atrasos de comunicação e é não-preemptivo, mas trabalha com grafos acíclicos genéricos. Seu funcionamento é semelhante ao da técnica HLF. Devido ao seu critério de escalonamento, HLFET também é conhecido como CP (*Critical Path*) [KN84].

4.4.3 *Highest Level First with No Estimated Times*

A técnica HLFNET [ACD74] é semelhante à HLFET, exceto por não possuir as estimativas dos tempos de execução das tarefas. Sua rotulação é realizada de forma idêntica à da HLF, considerando também que os tempos de execução são idênticos.

4.4.4 *Random List Schedule*

Neste caso a lista de tarefas executáveis não é ordenada. O único critério aceito é o respeito às dependências. Não são considerados os atrasos de comunicação, nem os tempos de execução das tarefas, e não é permitida preempção [ACD74].

Seus resultados são utilizados, em geral, como base em *benchmarks*.

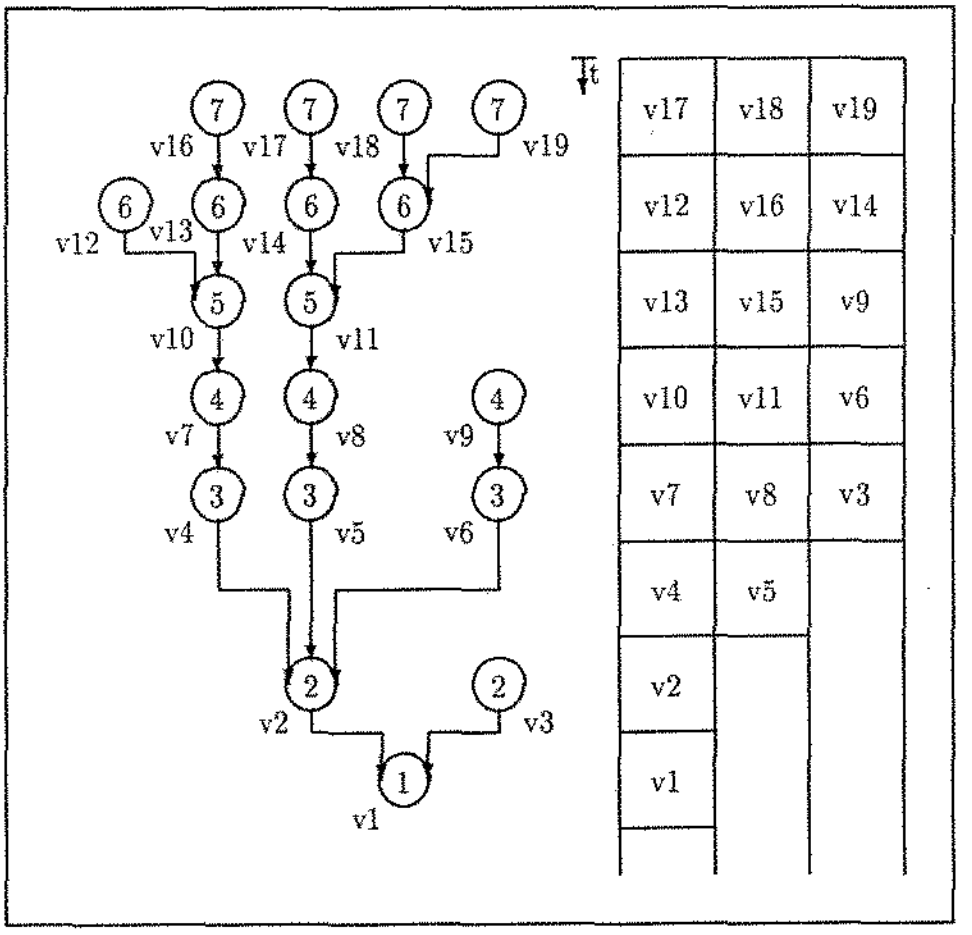

Figura 4.10: Exemplo da técnica de escalonamento HLF. Grafo rotulado e *Gatt chart* do escalonamento dos vértices em três UPs.

4.4.5 *Smallest Co-levels First with Estimated Times*

Na técnica SCFET [ACD74], a prioridade das tarefas corresponde ao negativo de seu co-nível. De resto, SCFET é semelhante a HLFET.

4.4.6 *Smallest Co-levels First with No Estimated Times*

A técnica SCFNET [ACD74] é semelhante à HLFNET, exceto pelo seu critério de prioridade, que é igual ao da técnica SCFET.

4.4.7 *Subset Schedule*

Esta técnica [MC69] trabalha com grafos de tempo de execução unitário e escalonamento preemptivo. Numa arquitetura com duas UPs, ela obtém resultados ótimos. Com mais de duas unidades, ela não garante tal eficiência.

O grafo é subdividido em grupos disjuntos, *subset sequences*. Cada grupo é constituído de vértices que possuem o mesmo nível.

A opção pela preempção desta técnica se baseia no fato de que com ela é possível a execução de tarefas em menor tempo do que no escalonamento básico, como mostrou a Figura 4.2. Supões-se nesta abordagem que não existem custos na preempção, nem atrasos de comunicação.

4.4.8 *Most Immediate Successors First*

A técnica MISF tem como política de ordenação o número de filhos de cada vértice. O vértice que possuir mais filhos será alocado primeiro. Se vértices possuirem o mesmo número de filhos, a alocação será aleatória.

4.4.9 *Critical Path/ Most Immediate Successors First*

A técnica CP/MISF [KN84] baseia-se em HLFET e *Subset Schedule*. Assim, quando dois vértices estão no mesmo nível, a ordenação considera os números de filhos, seguindo a política de ordenação de MISF.

4.4.10 *Large-Grain Dataflow*

Esta técnica [Bab84] leva em consideração os atrasos de comunicação, sem, no entanto, explorar todo o paralelismo do programa [KL88]. Este é dividido com o objetivo de que o tempo de execução de cada grupo de tarefas seja maior do que o tempo de comunicação de cada tarefa do grupo.

4.4.11 *Insertion Scheduling Heuristic*

ISH [KL92a] constitui um aperfeiçoamento de HLF em basicamente dois itens. O primeiro é a consideração dos atrasos de comunicação. E o segundo, a utilização dos tempos ociosos das UPs através do contínuo acompanhamento destes e da lista das tarefas habilitadas à execução.

4.4.12 *Duplication Scheduling Heuristic*

DSH [KL88, KL92a] é o aperfeiçoamento de ISH pela duplicação de tarefas para a redução dos custos de comunicação (ver Figura 4.8), o chamado conceito de duplicação de tarefas[8]. Este conceito foi usado pela primeira vez no escalonamento de tarefas em DSH.

4.4.13 *Mapping Heuristic*

MH [ERL90, ERL92] aborda o *Mapping Problem* (MP) [Bok81] analisando as contenções[9] da máquina. MP é o problema de alocar as tarefas às UPs considerando os atrasos de comunicação. MH tem a técnica HLF como base de funcionamento.

4.5 CP/MISF/∞

Com base no funcionamento da MFDM, foi desenvolvida no âmbito deste estudo a técnica de escalonamento de tarefas CP/MISF/∞. Seu objetivo é melhorar o desempenho do sistema através da alteração da ordem dos pacotes no anel da máquina. Neste processo, foi dada especial atenção ao tratamento dos laços dos programas.

Nas seções seguintes, CP/MISF/∞ é descrita em detalhes. Primeiramente, são abordados alguns aspectos de projeto. Em seguida, detalha-se seu funcionamento. E, por fim, são analisados alguns aspectos de implementação e sua atual implementação em gMDMS 2.2.

4.5.1 Aspectos da Nova Política de Escalonamento

A política FIFO utilizada na ordenação da lista de escalonamento da MFDM é uma das mais simples, mas esta não se demonstrou satisfatória. Dentre as várias técnicas de escalonamento disponíveis, optou-se por permanecer com a de lista de escalonamento devido aos seguintes fatores:

[8] *Task-duplication concept.*
[9] *Contention.*

- Sua concepção simples facilita sua implementação.
- Seu funcionamento por demanda suporta a incerteza do grafo, uma característica inerente à dinâmica dos laços.
- Sua característica de alocação "faminta" reflete satisfatoriamente os princípios do MFD. Como, devido aos limites físicos, não é possível executar todos os vértices habilitados, então que seja executado o maior número possível.

A nova política considera que:

- O grafo do programa já passou por um "empacotador", ou seja, os atrasos de comunicação foram analisados e o grafo remodelado [Vis].
- São conhecidas as estimativas dos tempos de execução das tarefas. Este tipo de informação é uma das saídas do "empacotador".
- Os laços não são aninhados numa única tarefa, possibilitando uma maior exploração dos vários tipos de paralelismo existentes (*doall, doacr, forall, doserial loops*, etc [PK87]). Isto impossibilita um escalonamento exclusivamente do tipo estático, pois não é possível determinar com precisão quantos vértices existirão no traço do programa, antes de sua execução.

4.5.2 Rotulação das Tarefas

A partir do momento em que existirem mais tarefas para serem processadas do que UPs disponíveis, estará formada uma lista de tarefas habilitadas. Com base em quais informações esta lista será ordenada?

Os vértices dos caminhos críticos do grafo têm preferência sobre demais, para que o processamento possa ser otimizado. Já que são conhecidas as estimativas dos tempos de execução das tarefas, a política empregada baseia-se em HLFET, que apresenta o melhor desempenho para grafos acíclicos genéricos [ACD74]. Mesmo assim, serão necessárias algumas adaptações otimizar a ordenação e o tratamento dos laços.

CP/MISF

As tarefas são rotuladas, definindo as prioridades para ordenação da lista. O rótulo de cada tarefa é composto pelo seu nível e o número de tarefas que a sucedem imediatamente (ver Figura 4.11).

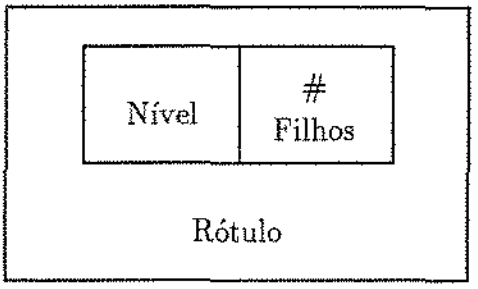

Figura 4.11: Rótulo das tarefas, para definição das prioridades na lista de escalonamento.

O nível de uma tarefa é igual ao comprimento do maior caminho entre a tarefa terminal e ela, inclusive. O comprimento desse caminho crítico é determinado sem levar em conta as possíveis iterações dos laços, como se o grafo fosse acíclico, simplesmente acumulando-se as estimativas dos tempos de execução das tarefas (ver Figura 4.12).

Tratamento dos Laços (∞)

Como os laços não são empacotados numa única tarefa, como normalmente ocorre na maioria das técnicas de escalonamento, deve ser criado um critério para o escalonamento das tarefas correspondentes. As referências [PK87, Tow86] são exemplos de trabalhos que também fazem tratamento especial de laços.

Pode-se ter uma expectativa do número de iterações de um laço ou até o número exato, como no caso de estruturas do tipo "faça x vezes". Entretanto, o controle individual de cada tarefa pertencente a um laço seria muito oneroso. Por isso, não supõe se conhecer estimativa alguma do número de iterações de qualquer tipo de laço.

Os laços são identificados num grafo orientado pela existência de ciclos (ver Figura 4.13(a)). Numa analogia à replicação de laços que ocorre no modelo de demanda[10], pode-se imaginar o grafo de um laço como um grafo acíclico, com um número infinito de replicações do trecho cíclico (ver Figura 4.13(b)).

Considerando-se essa analogia, rotular uma tarefa pertencente a um laço eqüivale a atribuir-lhe o nível ∞. Por conseqüência, todas as tarefas anteriores

[10] Este modelo de processamento paralelo utiliza o princípio de *lazy evaluation* [TBH82, Ash85, Kel85, Mag80].

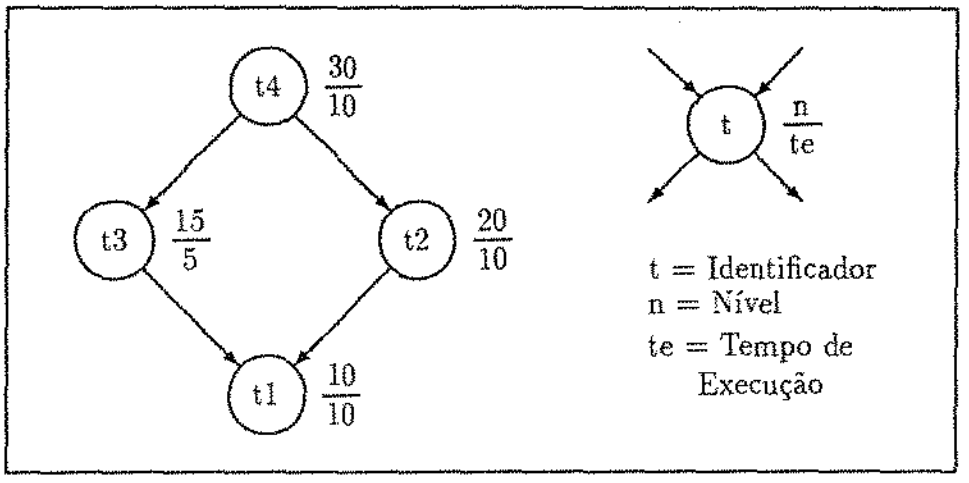

Figura 4.12: Exemplo de rotulação.

a um laço, com relação ao início do programa, teriam o nível ∞, o que acarreta um retorno funcional à política anterior. Contudo, dentro de um laço, todos os vértices teriam o mesmo nível, o que é incorreto em termos do critério caminho crítico.

Outra possibilidade é continuar gerando os níveis respeitando-se a característica cíclica dos grafos. Para ordenar as tarefas considerando-se a existência dos laços, seria incluído um campo no rótulo, para identificar os laços aos quais a tarefa pertence (ver Figura 4.14).

Seguindo a analogia com o modelo de demanda, toda tarefa deve ter seus laços sucessores contabilizados no seu campo de laços. Pelo mesmo motivo, vértices com maior grau de aninhamento de laços devem ter maior prioridade. É como se no desmembramento dos laços, um vértice mais interno tivesse um maior crescimento do seu nível, na remodelação do grafo, do que outro menos aninhado.

Para representar essas características, o campo de laço é preenchido por um polinômio. Em cada termo do polinômio, o expoente representa o grau de aninhamento e o coeficiente o número de laços com aquele aninhamento antecedidos pelo vértice. Por exemplo, um vértice com polinômio $2\infty^3 + \infty^1$, antecede dois laços com grau de aninhamento 3 e um de grau 1. Na contagem dos coeficientes incluem-se os laços dos quais o vértice considerado faça parte. Por exemplo, o vértice, do exemplo anterior pode ser parte do laço de grau 1.

No processo de geração dos polinômios, considera-se que o "empacotador"

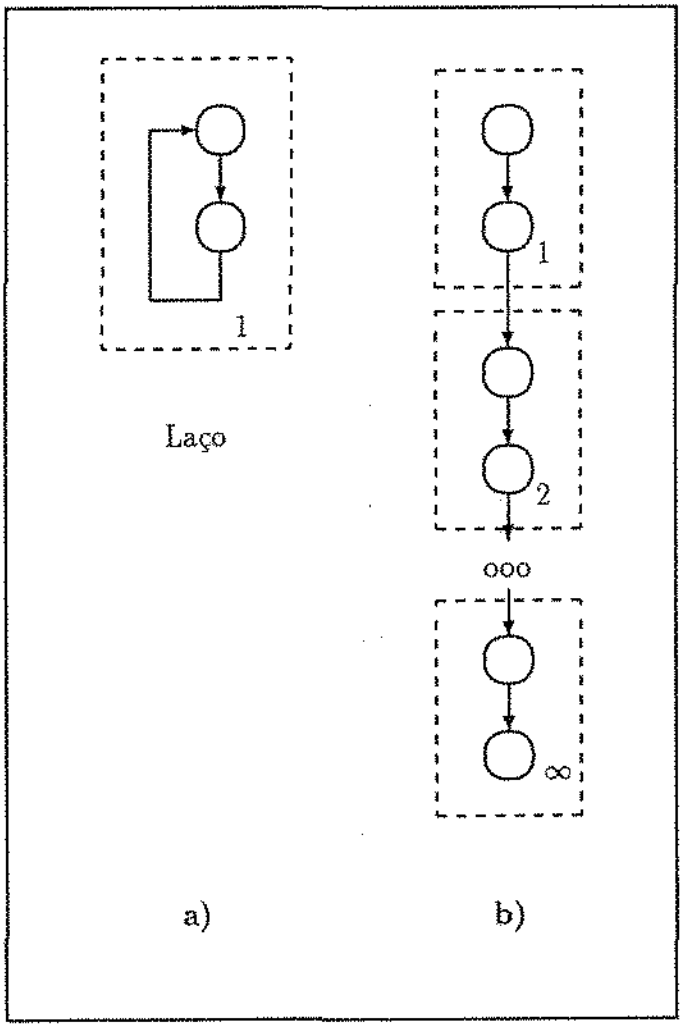

Figura 4.13: Visão do desdobramento de a) um laço em b) uma seqüência infinita do corpo do laço.

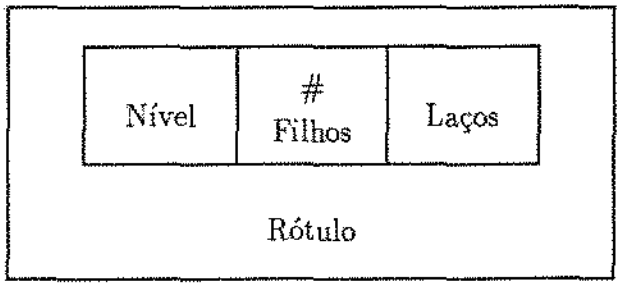

Figura 4.14: Rótulo das tarefas, com tratamento dos laços.

sinalizou os vértices de início e fim de cada laço. Com estas informações, torna-se trivial a determinação do grau do laço em que o vértice está contido. O polinômio de um vértice corresponde ao maior polinômio de seus filhos, supondo-se o grafo acíclico, e, caso este vértice esteja sinalizado como final de laço, será incrementado de 1 o coeficiente do termo associado ao grau corrente. O grau corrente é determinado pela diferença entre o número de vértices fim de laço e o número de vértices início de laço, até o momento.

O polinômio de rotulação não contém o termo ∞^0, que significa a relação do vértice a um laço de grau zero. Este termo pode ser, então, preenchido pelo nível do vértice, permitindo, assim, simplificar a representação (ver Figura 4.15).

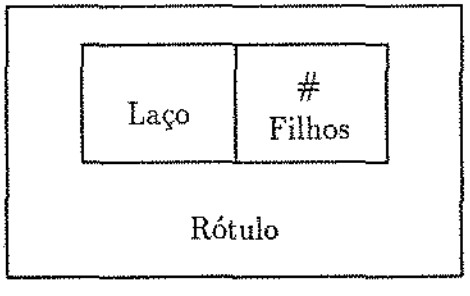

Figura 4.15: Rótulo simplificado das tarefas.

Ordenação

Tendo sido determinados os rótulos dos vértices (polinômio de laço e número de filhos), a lista de escalonamento é colocada na ordem ascendente dos rótulos. Ou seja, a tarefa com maior rótulo terá a maior prioridade. Define-se que um rótulo é maior que outro quando:

1. Seu polinômio de laço for maior.
2. Caso os polinômios de laço sejam iguais, seu número de filhos for maior.
3. Caso os polinômios de laço e os números de filhos sejam iguais, ele tiver chegado antes.

Essa política de escalonamento dá prioridade às tarefas com caminho crítico mais longo. Assim, otimiza-se o uso das UPs e de todos os recursos do anel, além de reduzir-se o tempo de execução do conjunto de tarefas T.

Na UX, essa política é implementada pelo árbitro de entrada. Contudo, é necessário , também, gerenciar a saída dos resultados, para se obter os resultados desejados.

4.5.3 Lista de Escalonamento do Árbitro de Saída

Ao contrário do árbitro de entrada, o árbitro de saída da UX tem que escalonar os resultados gerados pelas UPs num único ponto de saída. Em termos gerais, este tipo de problema é de simples solução. Entretanto, a ordem de liberação dos resultados interfere no acesso aos recursos do anel e na habilitação das tarefas. Portanto, é importante que a liberação dos resultados privilegie as tarefas de maior prioridade.

Com esse objetivo, o árbitro de saída deve dar preferência aos resultados da UP onde está alocada a tarefa de maior prioridade. Neste processo, não se deve alocar definitivamente a saída de resultados para a UP de maior prioridade num dado momento. É necessário reavaliar essas prioridades toda vez em que uma UP terminar a execução de uma tarefa.

Seguindo o mesmo raciocínio, as mensagens produzidas por uma tarefa devem ser escoadas numa ordem que também beneficie o sistema. Para tanto, em cada UP há uma lista de escalonamento com política semelhante ás anteriores. A mensagem que tiver como destino a tarefa com maior prioridade de execução terá prioridade de saída.

Assim, ficam coerentes as três soluções internas à UX. Toda lista formada na UX será tratada de acordo com as características de processamento da MFDM.

4.5.4 Preempção

Um aspecto muito importante no escalonamento de tarefas é a possibilidade de otimizar a execução das tarefas prioritárias recorrendo-se à preempção. No escalonamento básico, é possível que uma tarefa t_i seja iniciada antes que outra t_j de maior prioridade. A Figura 4.16(a) exemplifica um caso onde, mesmo seguindo, p. ex., a técnica CP/MISF, a tarefa t_6 inicia sua execução antes que t_3. Neste exemplo, as características de execução "faminta" da lista de escalonamento e do escalonamento dinâmico da MFDM levam a um desempenho aquém do ótimo, o qual é mostrado na Figura 4.16(b).

O escalonamento preemptivo possibilita, nesse caso, um melhor aproveitamento das UPs e um desempenho superior ao do escalonamento básico (ver Figura 4.16(c)). Como o escalonamento é dinâmico, não é possível reavaliar uma alocação. Apenas com base nas informações existentes na lista de escalonamento, é necessário definir a tarefa que vai ser colocada em execução, no momento da alocação. Então, aparentemente a preempção é benéfica para o problema em questão.

Por outro lado, interromper uma tarefa, salvar seu contexto, iniciar nova execução e restaurar o antigo contexto, não é simples e muito menos barato. A preempção tem um custo que não pode ser desprezado. Muntz e Coffman Jr. [MC69] defendem a preempção e alegam que seus custos não são consideráveis quando comparados com os seus benefícios.

A depender do tamanho médio da tarefa gerada pelo "empacotador" e dos recursos para troca e gerência de contexto, a preempção pode significar um *overhead* para a MFDM, gastando mais em trocas de contexto do que em processamento útil. Contudo, se as tarefas tiverem um tamanho expressivo e um sistema de preempção eficiente, o uso da preempção pode ser interessante.

4.5.5 Implementação Atual

Como ainda não foi realizada a integração do escalonador com o "empacotador", a técnica CP/MISF/∞ implementada em gMDMS difere da apresentada em alguns pontos. Entretanto, o principal conceito de CP/MISF/∞ é o tratamento dos laços. Como neste ponto a dependência do "empacotador" resume-se à sinalização do início e fim de cada laço, esta característica não sofre alterações.

Sem o "empacotador", a granularidade do grafo é ainda fixa e igual a uma instrução. Como, ao contrário de HLFET (CP), em gMDMS supõe-se que todas as instruções possuam o mesmo tempo de execução, adota-se HLFNET como base para a rotulação das instruções. Pela mesma razão, a preempção não é

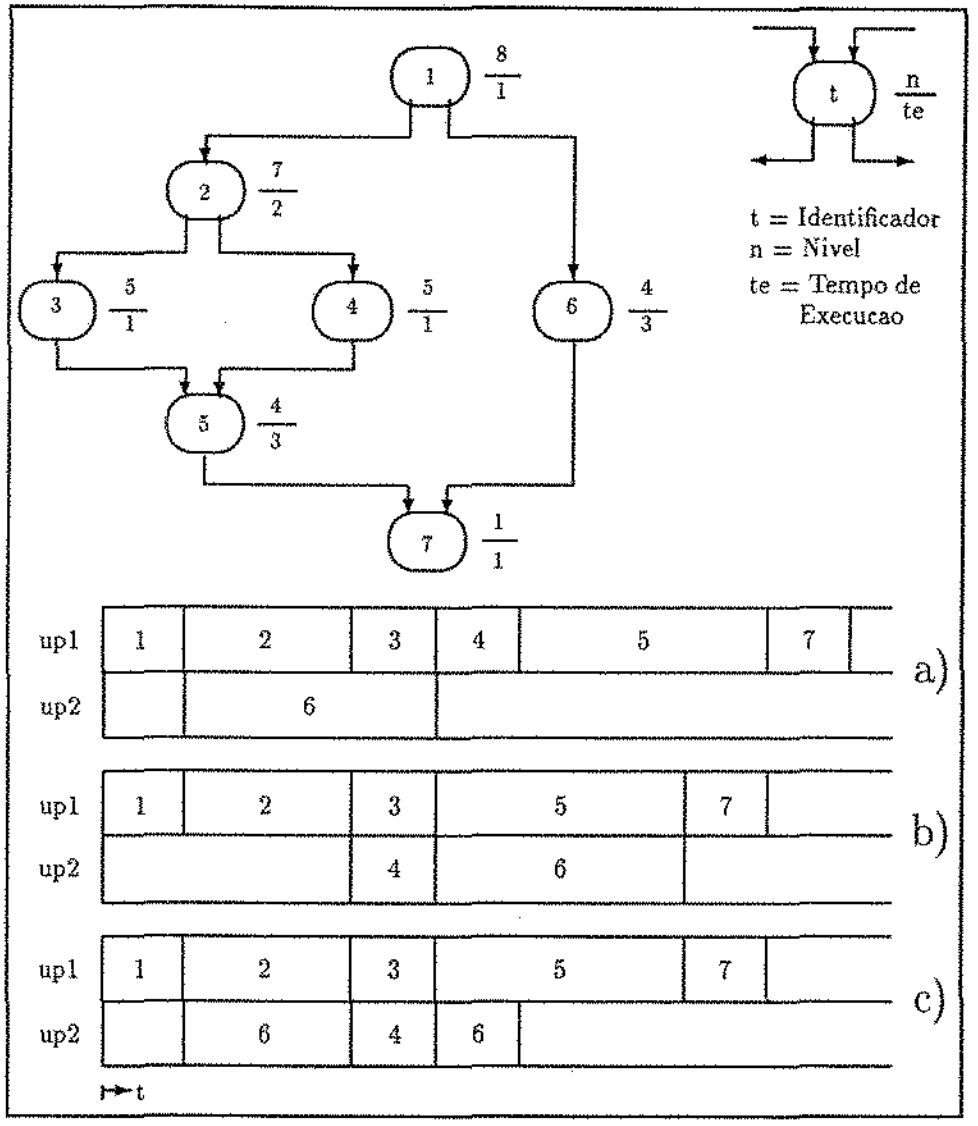

Figura 4.16: Comparação entre os escalonamentos a) CP/MISF/∞ sem preempção, b) ótimo, sem preempção, e c) CP/MISF/∞ com preempção.

habilitada.

Apesar dessas diferenças, a versão de CP/MISF/∞ implementada em gMDMS é suficiente para a análise do tratamento de laços (∞). A partir da integração do escalonador com o "empacotador", o impacto completo de CP/MISF/∞ no funcionamento da MFDM poderá ser avaliado.

Capítulo 5

Impacto do Escalonamento de Instruções

"As lembranças não têm preço."

"Sorte é aquilo que acontece
quando o preparo se encontra com a oportunidade."

A técnica FIFO utilizada na MFDM para o escalonamento de pacotes provoca uma super-utilização da UE. A ordenação dos pacotes pela ordem de chegada ocasiona a execução do grafo segundo uma política *breadth-first* e ocasiona o processamento das fichas de dados na UE na ordem em que foram geradas, não na ordem de consumo. Por isso, fichas têm que aguardar na UE até que possam ser encaminhadas às suas instruções destino.

Essa ordenação de pacotes tende a provocar momentos de baixo *throughput* na UE, o que afeta todo o anel. Esse procedimento também sobrecarrega o dispositivo de armazenamento da UE e acaba por causar a necessidade do uso da UT. A simples troca da política pode otimizar o uso das unidades da máquina, incluindo o do seu ponto fraco: a UE.

Este capítulo objetiva expor a estreita relação entre a ordem das fichas de dados no anel, o desempenho da UE e o desempenho do sistema como um todo. Adicionalmente, pretende-se mostrar que é possível obter melhores resultados substituindo-se a técnica de escalonamento FIFO adotada na MFDM. Não se pretende, no entanto, eleger uma técnica "ótima", nem alterar qualquer características ou aspecto estrutural do anel.

A seguir, são descritos os critérios de seleção das técnicas de escalonamento testadas e como elas foram implementadas em gMDMS. Nas duas últimas seções, analisa-se o impacto dessas técnicas, incluindo CP/MISF/∞, no funcionamento da MFDM.

5.1 Critérios de Escolha

É importante para a eficiência do sistema um perfeito ajuste entre a arquitetura alvo e a técnica adotada para o escalonamento de tarefas. A MFDM utiliza a técnica de lista de escalonamento com política de ordenação FIFO, que exibe algumas características desejáveis:

- Fácil implementação.
- Trabalha num processo "faminto" compatível com o princípio de *eager evaluation*.
- É dinâmica e não necessita de fase de compilação.
- Não depende do número de UPs disponíveis.

A escolha da técnica de lista de escalonamento deve ser uma conseqüência direta do modo de funcionamento da MFDM. Nela, todos os pacotes circulam num anel e são organizados em filas enquanto aguardam para serem processados. O acesso às unidades ocorre pela lógica do "próximo"; não existe qualquer requisito para seleção do próximo pacote.

Ao contrário do modelo von Neumann, onde a seleção da próxima tarefa a ser executada é conseqüência do fluxo de controle do programa, a seleção do próximo no MFD é conseqüência do fluxo dos dados. Na MFDM, qualquer pacote de uma fila poderia ser o próximo. Isto torna o sistema mais simples e dinâmico, pois não são necessários os dispositivos de controle de execução usados nas máquinas von Neumann. Contudo, a inexistência desses dispositivos inviabiliza o uso de técnicas de escalonamento mais elaboradas.

Como não se desejava a adicionar novas unidades à arquitetura original, optou-se por manter o uso da técnica de lista de escalonamento. Com isso, a especialização da política de ordenação será utilizada para analisar o impacto do escalonamento das tarefas na MFDM.

Esta decisão de manter a técnica de lista de escalonamento preserva as características FIFO descritas anteriormente exceto pela necessidade de uma fase

de compilação. As tarefas são rotuladas com base nas funções-objetivo de cada política. Esses rótulos serão a base de ordenação dos pacotes nas filas da MFDM.

Como a MFDM é um *pipe* circular é necessário aplicar a ordenação dos pacotes em todo o percurso do anel. A ordenação limitada à UX restringe o efeito da política de escalonamento a grupos de tamanho máximo igual ao número de UPs disponíveis. No resto do anel FIFO é usada implicitamente (ver 3.3).

5.2 Implementação em gMDMS

Como exposto em 2.2.2, é possível alterar os critérios de ordenação de pacotes nas filas de gMDMS. Esta ordenação baseia-se na rotulação prévia do grafo do programa. Cada instrução possui um rótulo que define sua prioridade de processamento relativamente às outras instruções.

Entretanto, como o que basicamente circula pelo anel são fichas de dados, estas também possuem um campo de rótulo. E, como a importância do dado no escalonamento é relativa à importância da instrução da qual ela é operando, as fichas de dados possuem como rótulo o rótulo da sua instrução destino, a qual é sempre única.

Na inclusão de um pacote numa fila, ele é posicionado de acordo com seu rótulo em ordem crescente dentre os pacotes já existentes. Analogamente, os pacotes são selecionados para processamento, ou seja, são excluídos das filas, a partir do de maior rótulo. Assim, o pacote de maior prioridade é processado nas unidades em primeiro lugar.

5.3 MFDM sem FIFO

A seguir, SUM20 é simulado com as políticas de ordenação MISF, HLFNET e CP/MISF. São usados os mesmos tipos de gráficos das simulações com FIFO (ver 3.3), possibilitando, assim, a comparação dos resultados e a análise do impacto de cada política no funcionamento da MFDM.

Devido às grandes diferenças observadas nos valores médio e máximo da utilização da memória da UE entre as diferentes políticas, os gráficos destas medidas têm diferentes valores de escala. Esta decisão possibilita a visualização clara das curvas, o que não seria possível de outra forma.

5.3.1 MISF

Como primeira alteração na ordem dos pacotes no anel, adota-se a simples priorização das instruções com maior número de resultados. *Most Immediate Successors First* (MISF) (ver 4.4.8) busca uma abertura mais rápida do paralelismo do programa. A ordenação das instruções com a mesma prioridade é realizada via FIFO.

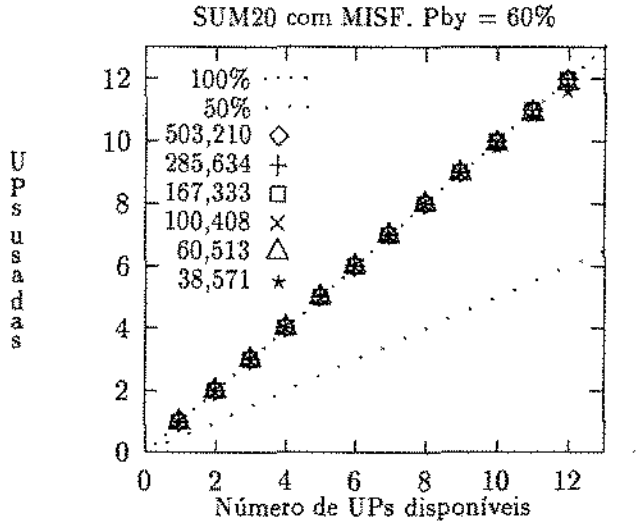

Figura 5.1: Curvas de desempenho do programa SUM20 com MISF, para diferentes níveis de PM.

O gráfico da Figura 5.1 mostra as curvas de desempenho de SUM20 simulada com MISF. Observa-se, em decorrência da simples troca da ordem dos pacotes no anel, uma redução da perda de desempenho registrada em FIFO.

As Figuras 5.2 e 5.3 mostram respectivamente os gráficos de utilização média e máxima da UE. Os níveis de ocupação média e máxima se apresentam tão altas quanto nos de FIFO. Contudo, como se observa na curva de utilização da UE de SUM20 com PM igual a 38,571 (ver gráfico da Figura 5.4), existem mudanças na distribuição das operações da UE.

O gráfico da Figura 5.5 mostra a curva de utilização das UPs de SUM20 com PM igual a 38,571. Observa-se uma melhora na utilização das UPs; não mais existem momentos de ociosidade de todas as UPs disponíveis, mas ainda existem momentos de ociosidade de algumas delas.

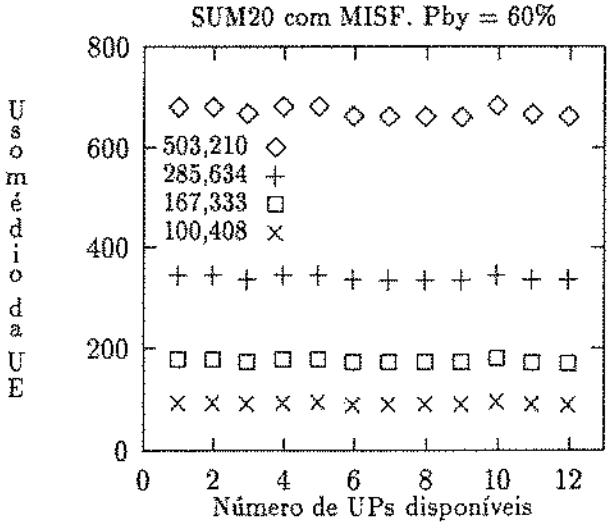

Figura 5.2: Curvas da utilização média da memória da UE para o programa SUM20 com diferentes níveis de PM e com a técnica de escalonamento MISF.

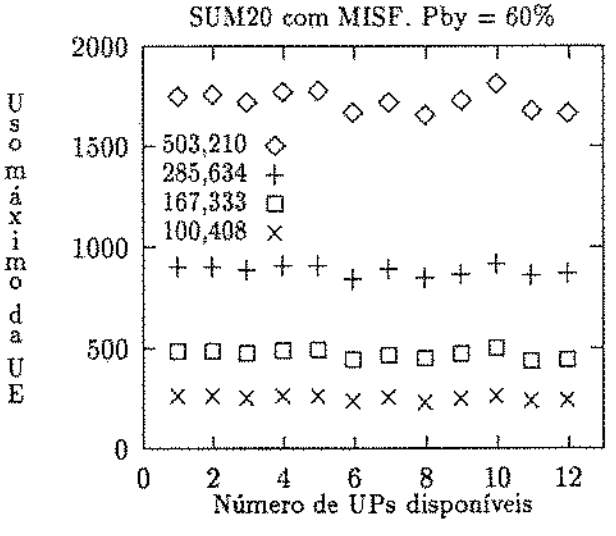

Figura 5.3: Curvas da utilização máxima da memória da UE para o programa SUM20 com diferentes níveis de PM e com a técnica de escalonamento MISF.

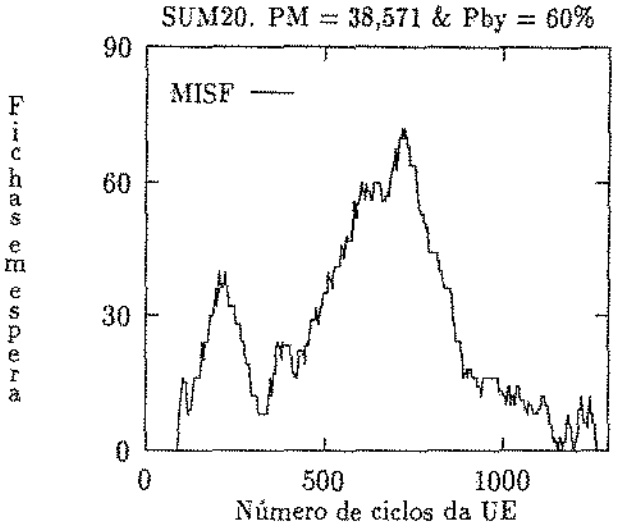

Figura 5.4: Curva de utilização da UE na execução com 12 UPs disponíveis de SUM20 com PM igual a 38,571 e com a técnica de escalonamento MISF.

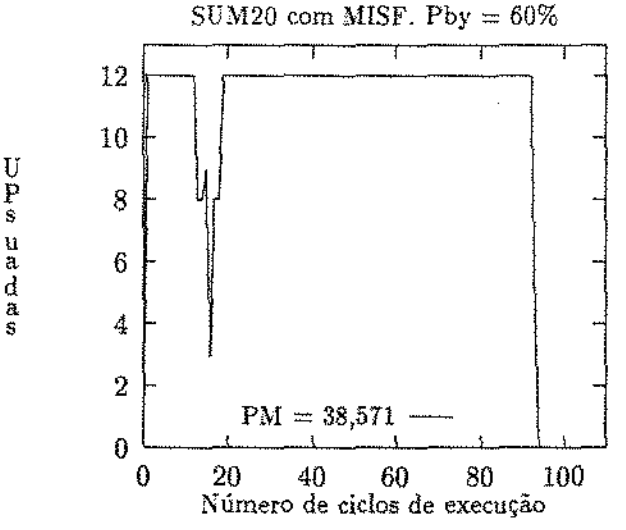

Figura 5.5: Curva de utilização das UPs, UX, na execução com 12 UPs disponíveis de SUM20 com PM igual a 38,571 e com a técnica de escalonamento MISF.

5.3.2 HLFNET

O objetivo básico da computação paralela é a execução mais rápida dos programas. Segundo esta idéia, deve-se concentrar os esforços no caminho mais demorado do código. Ou seja, deve-se priorizar o caminho crítico do grafo.

Como se admite que as instruções tenham o mesmo tempo de execução (ver 2.2.2), adota-se a política de ordenação HLFNET (ver 4.4.3). Com ela, espera-se avaliar os efeitos da troca da ordem de execução *breadth-first* de FIFO pela *limited-breadth*; executa-se *breadth-first* até a máquina estar totalmente ocupada e *depth-first* daí em diante [RS87]. Assim, como os pacotes de efeito mais próximo terão prioridade nas unidades, espera-se uma redução da ocupação da memória da UE. E, por conseguinte, espera-se evitar concentrações de operações *wait* e concentrações de operações *match* (ver 3.2).

A Figura 5.6 mostra o grafo de SUM rotulado com base em HLFNET. E o gráfico da Figura 5.7 mostra as curvas de desempenho de SUM20[1] simulado com HLFNET. Observa-se que não há perda de desempenho.

Os gráficos das Figuras 5.8 e 5.9[2] mostram respectivamente as curvas de utilização média e máxima da UE. A redução da utilização da memória da UE é enorme, comparada com FIFO e MISF. E existe uma coerente relação entre o número médio e máximo de fichas de dados que esperam emparelhamento e o número de UPs disponíveis. O gráfico da Figura 5.10 mostra a curva de utilização da UE de SUM20 com PM igual a 38,571. Não existem, como em FIFO e em MISF, picos de utilização. As operações estão distribuídas de forma mais mista pelo tempo de execução, gerando uma curva mais plana e uniforme. Resultados estes frutos da ordem de execução *limited-breadth*.

O gráfico da Figura 5.11 mostra a curva de utilização das UPs por SUM20 com PM igual a 38,571. Não há flutuações na utilização das UPs. Observe-se que, no final da execução, não são utilizadas todas as UPs disponíveis, uma vez que o número de instruções não é múltiplo do número de UPs. O valor de PMp (ver 2.1.4), no caso PM12 = 11,739, não expressa esse comportamento da curva de utilização das UPs. O que pode parecer uma perda de desempenho, é apenas um comportamento normal.

[1] Ressalta-se que SUM20 é a composição de vinte versões em paralelo de SUM (ver 3.3). Assim, a execução de SUM20 baseada em HLFNET é eqüivalente à execução de vinte versões em paralelo de SUM rotulada com base em HLFNET.

[2] Ressalta-se a alteração de escala dos gráficos.

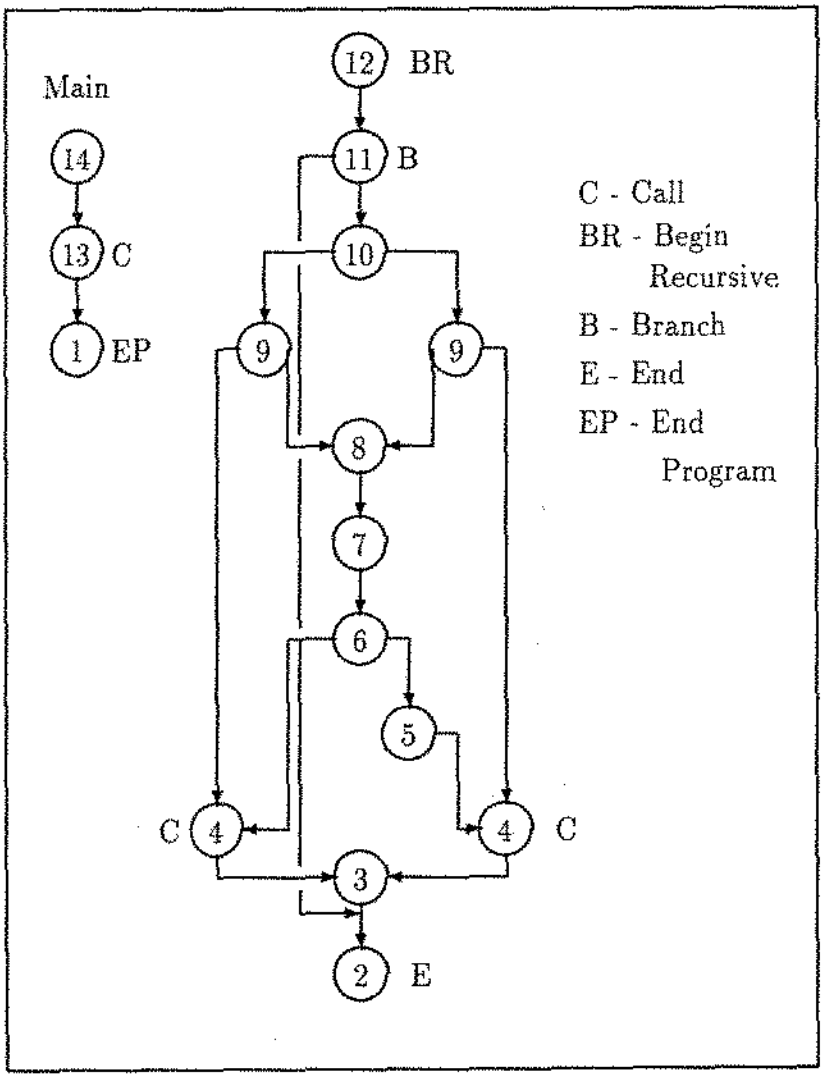

Figura 5.6: Grafo do programa SUM rotulado com base em HLFNET.

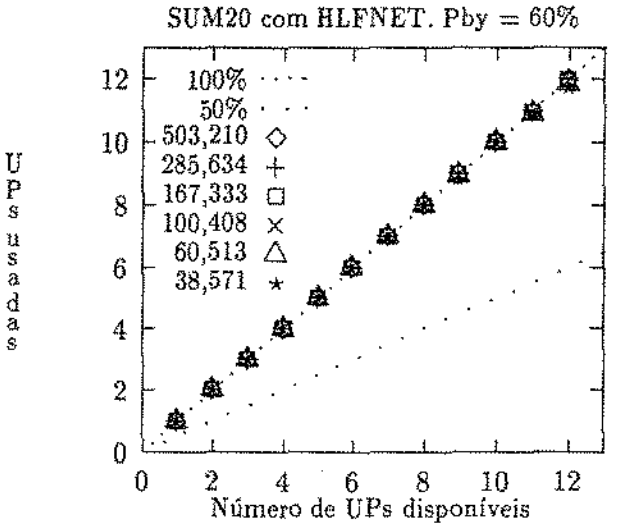

Figura 5.7: Curvas de desempenho do programa SUM20 com HLFNET, para diferentes níveis de PM.

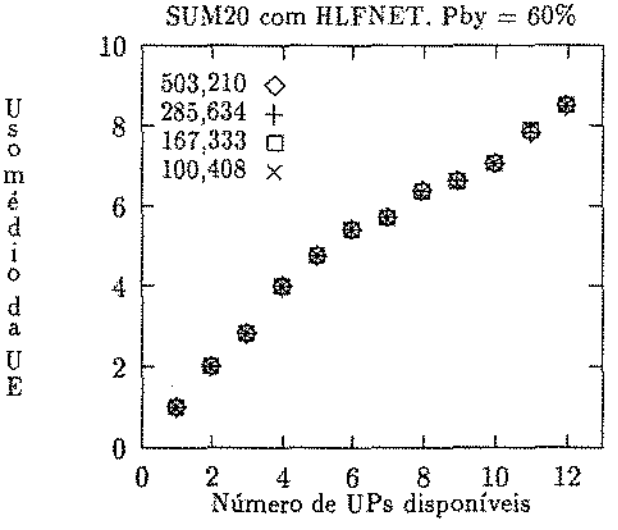

Figura 5.8: Curvas da utilização média da memória da UE para o programa SUM20 com diferentes níveis de PM e escalonamento HLFNET.

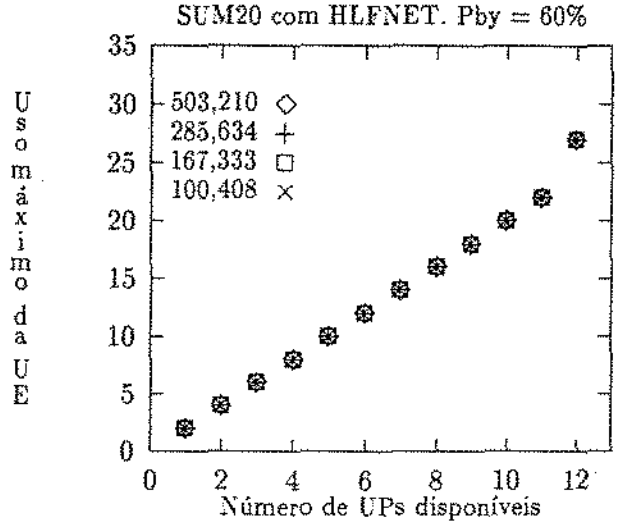

Figura 5.9: Curvas da utilização máxima da memória da UE para o programa SUM20 com diferentes níveis de PM e escalonamento HLFNET.

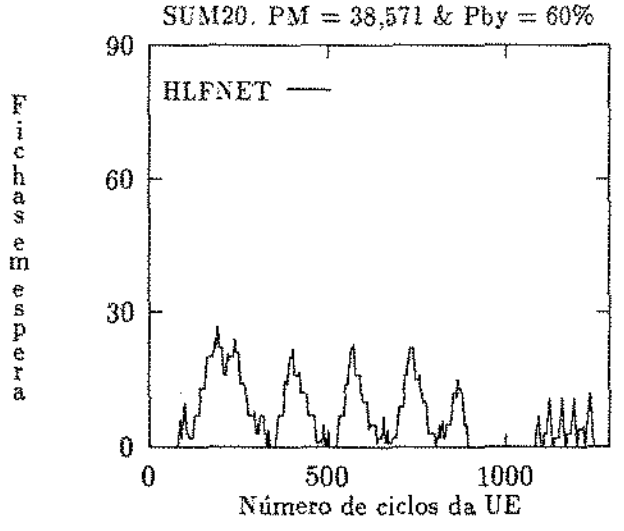

Figura 5.10: Curva de utilização da UE na execução de SUM20 com PM = 38,571, 12 UPs disponíveis e a técnica de escalonamento HLFNET.

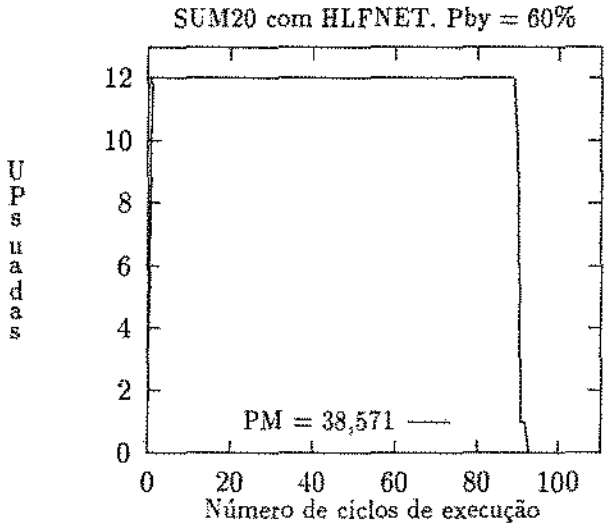

Figura 5.11: Curva de utilização das UPs na execução de SUM20 com PM = 38,571 com 12 UPs disponíveis e escalonamento HLFNET.

5.3.3 CP/MISF

A adoção da política de ordenação CP/MISF (ver 4.4.9) na execução de SUM20 apresenta os mesmos resultados de HLFNET. Como se considera que as instruções tenham tempos de execução iguais, CP se comporta como HLFNET. No caso de SUM20, pode-se observar que no grafo rotulado por HLFNET (ver Figura 5.6), as instruções de mesmo rótulo também possuem mesmo número de resultados. Assim, na solução dos conflitos de prioridade via CP, prevalece FIFO ao invés de MISF.

5.3.4 Análise dos Resultados

Como a MFDM é um anel simples, a ordem dos pacotes é um fator básico para o bom funcionamento do sistema. A busca do máximo paralelismo realizada por FIFO sobrecarrega a UE com fichas de longo tempo de permanência. Isto provoca a formação de seqüências de operações que degradam o *throughput* da UE.

O gráfico da Figura 5.12 resulta da sobreposição das curvas de utilização da UE na execução de SUM20 com PM = 38,571, 12 UPs disponíveis e as políticas de ordenação de pacotes usadas anteriormente. É clara a melhora na utilização

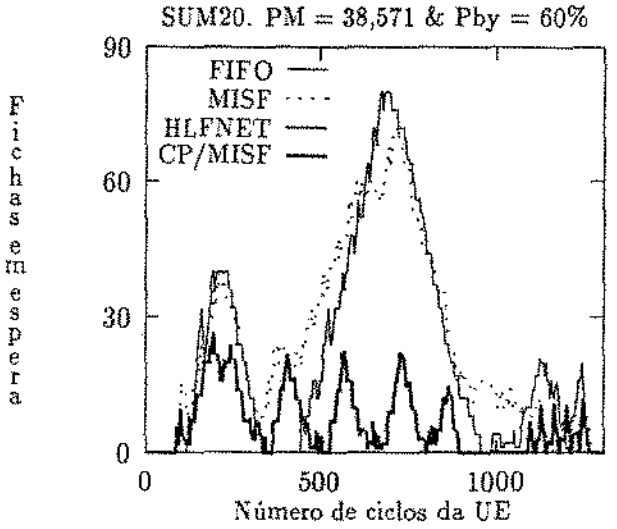

Figura 5.12: Curvas de utilização da UE na execução de SUM20 com PM = 38,571, 12 UPs disponíveis e diferentes políticas de ordenação dos pacotes no anel.

da memória da UE por HLFNET e CP/MISF. Esta melhora tem impacto direto no fornecimento de pacotes executáveis à UX.

Com HLFNET e CP/MISF, deixa de existir a perda de desempenho do sistema e há um drástico decréscimo do tamanho da memória necessária na UE. Por conseguinte, reduz-se proporcionalmente a probabilidades de ser necessário o uso da UT.

MISF não apresenta grande redução na utilização da memória da UE. Entretanto, como demonstram seus resultados, a troca da ordem dos pacotes possibilita uma melhora no fornecimento de pacotes à UX. Este resultado ressalta a delicada relação entre o desempenho da MFDM e a ordem dos pacotes no seu anel.

A Tabela 5.1 lista os tamanhos médios e máximos das filas das unidades na simulação de SUM20 com PM = 38,571 e 12 UPs disponíveis. Não existe sobrecarga em outro dispositivo de armazenamento da máquina quando do uso das outras políticas de escalonamento. A redução do uso da memória da UE com o uso de HLFNET e CP/MSIF e a não sobrecarga de outro dispositivo do anel são conseqüências da produção de fichas de dados gerenciada pela respectiva demanda.

Tabela 5.1: Exemplo da ocupação das filas de gMDMS.

		FIFO	MISF	HLFNET	CP/MISF
fMU	TamMax†	91	45	39	39
	TamMed‡	19,607	15,822	16,013	16,013
fISU	TamMax	30	27	23	23
	TamMed	7,329	7,315	7,467	7,467
fEU	TamMax	80	76	80	80
	TamMed	37,298	35,745	42,858	42,858

(† Tamanho Máximo. ‡ Tamanho Médio.)

5.4 MFDM com CP/MISF/∞

Como SUM20 não possui laços, sua execução com CP/MISF/∞ (ver 4.5) comporta-se tal como CP/MISF. Ou seja, seus resultados são idênticos aos de HLFNET. Assim, para visualizar o impacto de CP/MISF/∞ é necessário o uso de um código com laços.

A Figura 5.13 mostra uma composição envolvendo SUM: SUML. Nela, duas chamadas de SUM são realizadas em paralelo, estando uma delas inserida num laço, bloco I. Esta estrutura possibilita uma rotulação baseada em CP/MISF simétrica em relação aos blocos. Contudo, o rótulo do bloco II, que não está inserido num laço, tem grau menor, $(\infty + l)$, do que o do bloco I, $(\infty^2 + \infty + l)$. Assim, é possível ter uma idéia do impacto de CP/MISF/∞ no funcionamento da MFDM.

Como SUM, SUML também possui trechos com baixo paralelismo (ver Figura 3.9). A mesma solução é então aplicada: 20 versões em paralelo de SUML constituem SUML20. E, para avaliar o comportamento de CP/MISF/∞, três casos de SUML20 são analisados.

A Tabela 5.2 descreve a relação entre os blocos de SUML20 em termos do contador das instruções *Branch* (ver 2.2.4) do código de SUM (ver Figura 3.16 ou 5.6) dos blocos. O caso 1 possui os blocos simétricos, sendo que o laço do bloco I só interfere na rotulação do grafo. No caso 2, o bloco I possui Sinf três vezes maior do que o do bloco II. E, no caso 3, o bloco II possui PM e Sinf ambos maiores do que os do bloco I. Em todos os três casos, o Pby é por volta de 62%.

Como CP/MISF/∞ privilegia o bloco I, os casos 1 e 3 não lhe são favoráveis. Já o caso 2 possui uma coerência com a rotulação por ∞. Da mesma forma, o

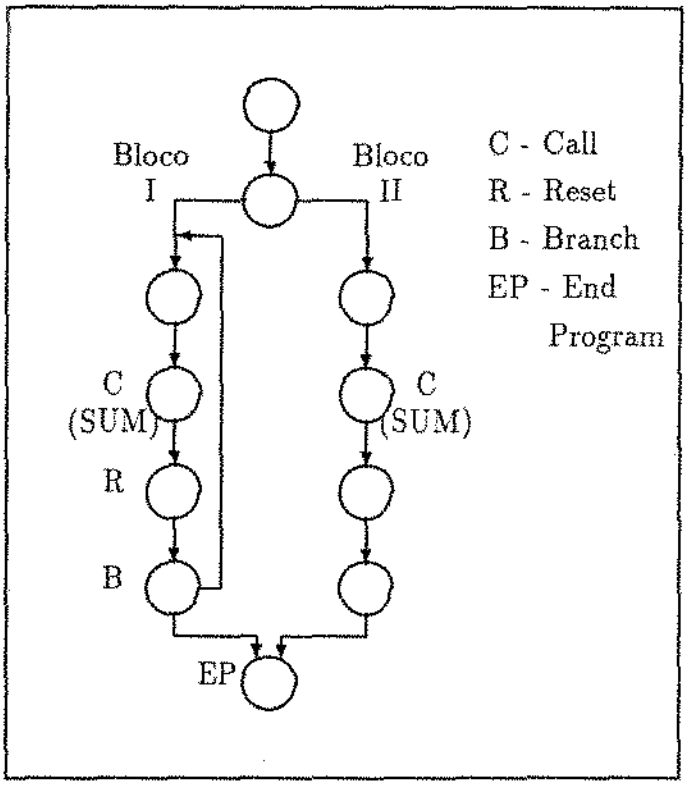

Figura 5.13: Grafo do programa SUML.

Tabela 5.2: Descrição dos casos de SUML20.

	# *Branch*		PM	# iterações
	Bloco I	Bloco II	SUML20	do bloco I
Caso 1	60	60	70,000	0
Caso 2	60	60	35,297	2
Caso 3	60	140	81,860	0

caso 1 é favorável a rotulação por HLFNET e CP/MISF.

Os gráficos das Figuras 5.14, 5.15 e 5.16 mostram, respectivamente, as curvas de desempenho dos casos 1, 2 e 3 de SUML20 com diferentes políticas de ordenação de pacotes. Observa-se que MISF, HLFNET e CP/MISF possuem quase as mesmas curvas de desempenho em todos os casos. Dentre estas, apenas as do caso 2 apresentam uma visível perda de desempenho. Já as curvas de FIFO apresentam perda em todos os casos.

As curvas de CP/MISF/∞, indicadas abreviadamente por C/M/∞, apresentam sempre um desempenho visivelmente igual a 100%. Mesmo nos casos desfavoráveis, 1 e 3, CP/MISF/∞ apresenta os melhores resultados dentre as políticas de ordenação testadas.

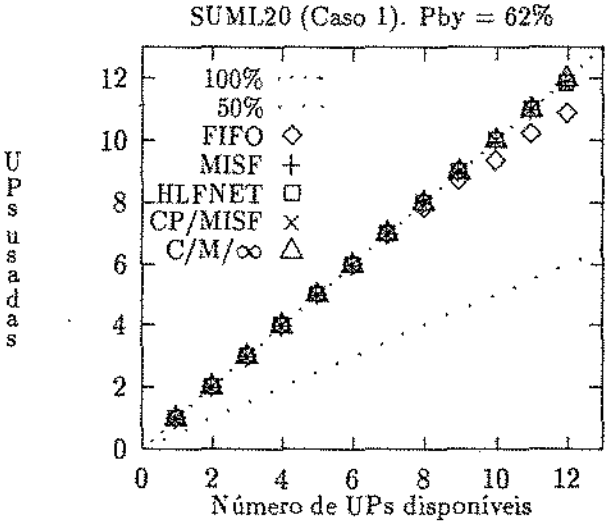

Figura 5.14: Curvas de desempenho do programa SUML20 (caso 1) para diferentes políticas de ordenação.

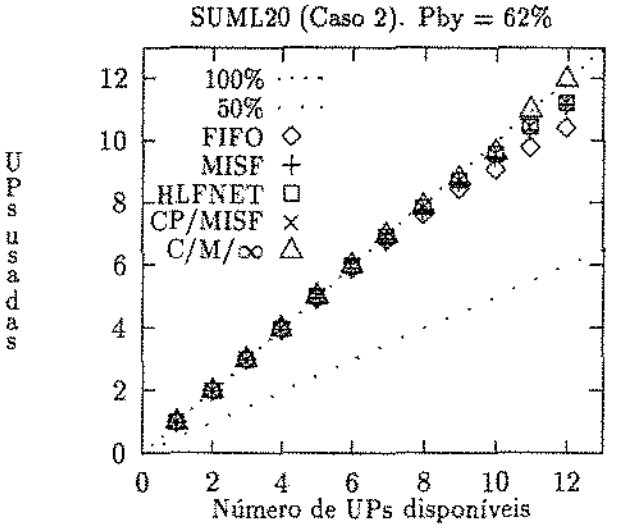

Figura 5.15: Curvas de desempenho do programa SUML20 (caso 2) para diferentes políticas de ordenação.

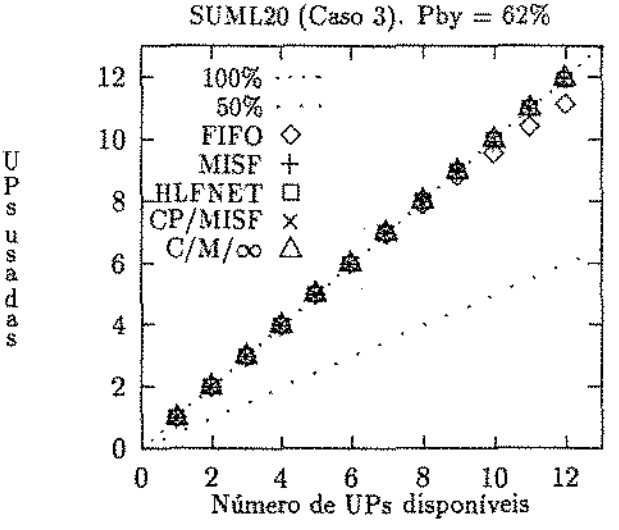

Figura 5.16: Curvas de desempenho do programa SUML20 (caso 3) para diferentes políticas de ordenação.

Capítulo 6

Conclusões

"Jamais corte o que pode ser desatado."

"Depois que você chega no topo da colina,
você ganha velocidade."

No capítulo 1, seção 1.6, foram descritas as metas desta dissertação. Dentre elas, o principal ponto era a análise do impacto do escalonamento de instruções no desempenho da MFDM, o que foi desenvolvido no anterior.

Aqui estão resumidas as principais conclusões deste estudo e destacadas suas principais contribuições. Ao final, são sugeridas algumas oportunidades para futuros desenvolvimentos.

6.1 O Impacto do Escalonamento de Instruções

Na busca do aumento da capacidade computacional, a computação paralela tem recebido atenção constante. Foram seguidos basicamente dois caminhos: a adaptação do modelo convencional de von Neumann e o desenvolvimento de modelos originalmente paralelos. Neste último caso, dois modelos merecem destaque: o de fluxo de dados (ver 1.4) e o de demanda.

Como o modelo de von Neumann é originalmente seqüencial, sua remodelação para funcionamento paralelo apresenta uma série de dificuldades. Entretanto, por ser largamente conhecido e difundido, o modelo de von Neumann mereceu grande atenção número de trabalhos. Atualmente, arquiteturas paralelas baseadas nesse

modelo mostram-se eficientes para o processamento de código cuja execução segue um padrão regular. Dentre estas, destacam-se as arquiteturas vetoriais.

No MFD, a MFDM (ver 2.1) representou um dos esforços mais importantes e um dos que mais contribuíram para o desenvolvimento dos conceitos deste modelo. Entretanto, mesmo possuindo uma base teórica coerente, as implementações do MFD também apresentaram problemas e limitações.

Como essas limitações variam de um modelo para outro, verifica-se atualmente um movimento no sentido de combinar aspectos favoráveis em soluções híbridas: arquiteturas híbridas[1]. Entre esses trabalhos, pode-se citar [Kam94, NBM93, BI92, NA89, BE87, Sow87].

A MFDM apresenta basicamente dois grandes problemas: perda de desempenho e sobrecarga da UE. A UE tem sido apontada como a causa da perda de desempenho, o "gargalo" do sistema. Entretanto, como se mostrou no capítulo 3, mesmo sendo a UE um ponto "delicado" da máquina, nem sempre ela é a responsável pela perda de desempenho descrita nas publicações afins.

A MFDM é um anel seqüencial através do qual circulam pacotes contendo dados. Considerando que a UC só se comunica com o H no início e no final das execuções, a UE é a única unidade no anel que pode possuir uma taxa de produção de pacotes inferior à de consumo. Esta variação na taxa de produção da UE pode provocar "bolhas" no *pipe*, ou seja, pode provocar a ociosidade das unidades seguintes por falta de oferta de pacotes.

De modo geral, o paralelismo no sistema é ampliado na UX, quando da execução de instruções de dois resultados. E esse paralelismo é reduzido na UE na chegada dos primeiros pacotes destinados a instruções de dois operandos. Esses pacotes requisitam operações *wait* e as fichas de dados são armazenadas à espera dos respectivos pares.

As operações que manipulam a memória da UE caracterizam-se por serem lentas. Adicionalmente, concentrações de ocorrências destas operações, especialmente *wait*, degradam a taxa de produção de pacotes da UE, e a tornam um "gargalo" no sistema.

Entretanto, a medida Pby não expressa a existência dessas concentrações na execução dos programas. Obviamente programas com Pby = 100% não têm problemas com a UE, pois só possuem instruções de um operando. Assim como, quanto maior o valor de Pby, maior a probabilidade de ocorrerem essas concentrações. Fora isso, nada mais pode ser deduzido desta medida isoladamente. Pby é um valor médio que não expressa o comportamento das operações da UE.

Da mesma forma, PM também é uma medida média e não expressa a dis-

[1] *Hibrid architectures.*

tribuição do paralelismo pelo código do programa. Tanto PM quanto Pby, são medidas auxiliares importantes para avaliação dos programas, mas seu uso deve vir em conjunto com outras informações. Como as análises da MFDM consideraram as medidas Pby e PM como medidas base, as análises tiveram seus resultados distorcidos.

O código de alguns programas possui trechos protegidos pelas dependências de dados que geram concentrações de operações que manipulam a memória da UE. Somente através da alteração do código é possível evitar os efeitos destes trechos na UE. O grupo de Programas-Constantes, descrito na seção 3.1.1, contém alguns exemplos dessa classe. Programas com essas características não são eficientemente executados pela MFDM. Somente através de modificações na arquitetura original da máquina é que estes trechos de código poderão ser adequadamente processados.

Já outros programas não possuem uma distribuição constante do paralelismo, como, p. ex., os Programas-Recursivos (ver 3.1.2). A MFDM é competente para a execução desta classe de programas, mas, nos momentos de baixo paralelismo, observa-se uma sensível perda de desempenho.

Outros programas têm paralelismo constantemente alto, mas podem ter problemas na UE, dependendo da ordem em que seus dados circularem pelo anel. Como as operações que manipulam a memória da UE são operações delicadas, seu uso deve ser bem administrado. Entretanto, a técnica FIFO usada para o escalonamento de pacotes na MFDM não realiza este controle. Pelo contrário, a ordem de execução *breadth-first* adotada por FIFO tem a tendência de antecipar as operações *wait*, provocando períodos de baixa taxa de produção de pacotes pela UE. Ou seja, FIFO tende a provocar o uso indevido da UE, tornando-a o "gargalo" do sistema. Em conseqüência da antecipação das operações *wait*, a memória da UE é super-utilizada, o que aumenta a possibilidade do uso da UT, degradando o desempenho das operações de acesso à memória. A conseqüência final desta cadeia é a perda de rendimento devida à falta de pacotes executáveis a serem processados pela UX em certos períodos.

A especialização da técnica de escalonamento da MFDM tem impacto direto no funcionamento e no desempenho das unidades da máquina. Os resultados das simulações com outras técnicas de escalonamento expostos no capítulo 5 demonstram esta delicada relação entre o comportamento do anel e a ordem dos pacotes. Por estas razões, recomenda-se a substituição de FIFO por uma técnica mais apropriada ao funcionamento da MFDM.

Dentre as técnicas testadas, destacam-se HLFNET e CP/MISF/∞, cujos pressupostos mostram-se mais compatíveis com o sistema. Com elas, observou-se um uso relativamente baixo da memória da UE, diminuindo o risco de recurso

à UT. Esta redução não implica no aumento do uso de outros dispositivos de armazenamento do sistema, em virtude do controle sobre a produção dos dados com base na respectiva demanda.

A partir desses resultados, pode-se concluir que FIFO é uma causa indireta da perda real de desempenho apresentada pela MFDM. Por isto, a especialização do controle sobre a ordem do fluxo dos dados pelo anel proporciona uma grande melhora na utilização das unidades da máquina e um conseqüente aumento do desempenho do sistema.

6.2 Contribuições

Esta dissertação é um dos primeiros trabalhos a reavaliar porque a perda de desempenho da MFDM. Neste processo, mostrou-se que as medidas Pby e PM são inadequadas como indicadores base de avaliação. Mostrou-se também que a UE realmente pode causar perda de desempenho no sistema. Seu *throughput* é sensível à ordem das operações requisitadas pelas fichas de dados. E, como as unidades da MFDM são organizadas num anel seqüencial, baixas taxas de *throughput* da UE podem causar a ociosidade de UPs e a conseqüente perda de desempenho da máquina.

Como a variação do *throughput* da UE é provocada pela ordem das fichas de dados no anel, é necessário gerenciar esse fluxo, de modo a manter níveis adequados de *throughput*. Por sua vez, a ordem das fichas de dados é o que define a ordem das execuções na MFDM. E, como definir a ordem das execuções é parte do problema de escalonamento de tarefas, o controle sobre o fluxo dos dados no anel faz parte do escalonamento de tarefas da máquina. Ou seja, o aperfeiçoamento da técnica de escalonamento de tarefas da MFDM pode aumentar o desempenho do sistema. A identificação desta relação e a análise do problema de escalonamento no MFD são pontos desenvolvidos nesta dissertação.

Para poder verificar o impacto da alteração da técnica de escalonamento sobre o desempenho da MFDM, foi desenvolvido gMDMS, um simulador que faz a modelagem do nível lógico da máquina.

As simulações realizadas permitiram comprovar o impacto direto da técnica de escalonamento adotada sobre o desempenho da UE e da máquina como um todo. A especialização do escalonamento traz consideráveis ganhos ao sistema, sem a necessidade da adição de novas unidades ao anel.

No processo de análise das técnicas de escalonamento, foi desenvolvida a técnica CP/MISF/∞. Ela resulta da integração de outras técnicas com o conceito de tratamento de laços (∞) desenvolvido nesta dissertação.

Embora a especialização do escalonamento da MFDM consiga em geral evitar as seqüências de instruções que degradam o *throughput* da UE, existem programas com trechos de código protegidos por dependências de dados que continuam provocando essas perdas na UE. Somente alterações nas características da máquina ou no código gerado poderão contornar este problema.

Durante o período de desenvolvimento deste trabalho, o autor participou do "IV Simpósio Brasileiro de Arquiteturas de Computadores – Processamento de Alto Desempenho", sediado em São Paulo, SP, em outubro de 1992, com apresentação e publicação nos anais do simpósio [LC92]. Neste evento foi apresentado um estudo da importância de se considerar os atrasos de comunicação no escalonamento de tarefas. Em 1993, através de convênio entre a UNICAMP e a Universidade de Gifu, Gifu, Japão, o autor foi contemplado com bolsa de estudos da organização japonesa Nakashima Zaidan para prosseguir pesquisa na Universidade de Gifu. O período da bolsa tinha duração inicial de um ano, abril de 1993 a março de '94, tendo sido estendida por mais um ano, abril de 1995.

Durante a permanência na Universidade de Gifu, o autor participar dos encontros regionais dos seis institutos de elétrica e demais engenharias, dos anos de 1993 e '94, com apresentação e publicação nos anais destes encontros [LGC93, LGC94]. Nestes eventos foram apresentadas análises do impacto do escalonamento de instruções e do conceito de *sequential blocking* no desempenho da MFDM, respectivamente.

Também abordando o impacto do escalonamento de instruções na MFDM, está em fase de avaliação o manuscrito "The Impact of Instruction Scheduling on the Speed of the MDFM" – Paulo Lorenzo, Munehiro Goto e Arthur J. Catto – na revista *Journal of the Brazilian Computer Society*. E, abordando as medidas de análise da MFDM, está também em fase de avaliação o manuscrito "A Note on the Soundness of the Performance Measures of the MDFM" – pelos mesmos autores – no VII SBAC-PAD, Simpósio Brasileiro de Arquitetura de Computadores – Processamento de Alto Desempenho.

Avançando na pesquisa descrita nesta dissertação, foi desenvolvido o conceito de *sequential blocking* e implementado no simulador gMDMS. A idéia básica deste estudo é o empacotamento de trechos seqüências de programa, sendo que os pacotes podem ter tamanho variado. Com base nos resultados obtidos e na análise destes, foi elaborado o artigo "Packing Sequential Stretches in the MDFM" – Paulo Lorenzo, Munehiro Goto e Arthur J. Catto – com publicação prevista para abril de 1995 pela revista *IEICE Transactions on Information and Systems*. Outras sugestões para trabalhos futuros são apresentadas na próxima seção.

6.3 Trabalhos Futuros

Como esta dissertação é um dos primeiros trabalhos que abordam o impacto do escalonamento de instruções na MFDM, procurou-se durante as linhas anteriores citar referências a outros possíveis caminhos dos seguidos. Assim, espera-se que em trabalhos futuros esta dissertação possa ser útil, tanto no esclarecimento de sua abordagem, como no acesso a outras.

6.3.1 *Petri Nets*

Dentre outras possíveis abordagens, pode-se destacar o uso da teoria de *Petri Nets*, devido ao isomorfismo existente entre GFD e uma classe de *Petri Nets* [KBB87], na abordagem do escalonamento das tarefas. Esta abordagem pode auxiliar também na modelagem de máquinas baseadas no MFD.

6.3.2 A Técnica de Escalonamento

Neste estudo analisou-se o impacto da técnica adotada para escalonamento de instruções no desempenho da MFDM. Para tanto, foram construídos programas-teste com características desejáveis para avaliar o sistema.

Como próximo passo, é necessária a simulação de programas clássicos. A análise das características mais comuns nesses programas deverá contribuir para a escolha da técnica mais adequada para escalonamento na MFDM.

A definição da nova técnica é o passo seguinte. Nesta dissertação, provou-se a importância da substituição da técnica original, mas não se apontou por qual. Para tanto, será necessário um estudo mais aprofundado da viabilidade de implementação de técnicas mais elaboradas na MFDM.

6.3.3 CP/MISF/∞

A técnica CP/MISF/∞ foi desenvolvida com base nas características da MFDM. No entanto, sua avaliação completa foi prejudicada pela não disponibilidade de um "empacotador" de código.

Para comprovar-se a eficiência de CP/MISF/∞, será necessário integrá-la com o "empacotador" e adaptar o simulador gMDMS ao conjunto. Com base em novos testes, poder-se-á verificar sua eficácia e a eficiência de seu conceito de tratamento de laços (∞). Em seguida, será necessária sua migração para outros sistemas e a análise de seu desempenho.

Apêndice A

gMDMS – Guia do Usuário

O objetivo deste simulador é a representação das execução de programas pela MFDM levando em conta o fluxo das fichas de dados pelo sistema e a ordem em que isso ocorre. Para analisar o impacto do escalonamento de instruções, é necessário medir o efeito das alterações na ordem de execução das instruções, ou seja, na ordem das fichas de dados.

Para isso, não é necessário conhecer o valor dos dados, nem os tipos de operações simples realizadas; basta identificar as instruções de controle e o aspecto do grafo do programa. É o mesmo que trabalhar na abstração que representa os dados como "bolas pretas" fluindo pelos "canos" do programa.

A seguir, apresenta-se um resumido guia do usuário do simulador gMDMS (Versão 3.1).

A.1 Utilização

Linha de comando (>):

>gMDMS [fileName [up1 [up2 [s]]]]

- >gMDMS – Além de simular a execução dos grafos, esta ferramenta também pode gerá-los. Para gerar os grafos, é só chamar o simulador sem parâmetros: >gMDMS. A fase de geração de novos grafos é explicada em A.4.
- >gMDMS fileName – Executa o grafo de nome "fileName".graph com 12 UPs disponíveis durante todo o tempo da execução.
- >gMDMS fileName up1 – Executa o grafo com up1 UPs disponíveis.

- >gMDMS fileName up1 up2 – Executa o grafo diversas vezes, com o número de UPs disponíveis variando de up1 até up2 (up1 $\leq$ up2).
- >gMDMS fileName up1 up2 s – Executa o grafo e gera um arquivo "fileName".shape, cujas linhas contêm os vértices executados em cada ciclo da UX. Se houver mais de uma execução (up1 $\neq$ up2), todas estarão neste arquivo. Em geral, só é interessante executar para um conjunto de UPs disponíveis ($up1 = up2$).
- Devido à falta de dados sobre a relação de tempo entre as unidades do sistema quando há mais de 12 UPs disponíveis, simulações com mais de 12 UPs são realizadas considerando-se a UE rápida o suficiente para processar todos os pacotes da fUE e a UAI, todos os pacotes da fUAI. Assim para identificar o PM e o Pmax de um grafo, basta executá-lo com UPs acima do máximo provável, verificando o limite da implementação.

A.2 Itens para Execução

O primeiro passo da simulação é a identificação da técnica de escalonamento que se deseja utilizar. As seguintes mensagens são apresentadas antes da simulação:

Policy (<0> FIFO <1> HLFNET <2> CP/Inf):

- FIFO – Seleciona o escalonamento original da MFDM.
- HLFNET – Seleciona o escalonamento HLFNET-CP.
- CP/Inf – Seleciona o escalonamento CP/∞ sem o conceito de MISF.

MISF?: (<0> No <1> Yes):

Nesta opção, caso *Yes* o conceito de MISF é utilizado. Ou seja, as fichas de dados com destino a instruções de dois operandos têm prioridade. Caso *No*, não há interferência.

Caso haja algum vértice do tipo *branch*, *consumer* ou *reset* cujo campo *counter* possui o valor zero, aparecerá na tela o número do vértice e será solicitado um valor positivo para o campo. Isto possibilita a execução de um mesmo grafo com diferentes repetições dos laços.

A.3 Resultados

A cada execução, diversas análises são mostradas na tela. Dentre elas o PMn, a Pby e os parâmetros de utilização da filas do sistema: fUAI, fUX e fUE. São utilizadas três casas decimais para os dados não-inteiros.

As mesmas informações mostradas na tela estão contidas no arquivo "fileName".res. À cada linha, são organizados os resultados das várias execuções. Informações sobre a ocupação da UE e do paralelismo durante a execução são encontrados, respectivamente, nos arquivos "fileName" com extensões .MU e .par. O *layout* destes arquivos é compatível com o *software* de geração de gráficos GNUPLOT.

A.4 Geração de Grafos

Na geração de novos grafos são requisitados o nome do grafo, sem extensão, e o número de instruções existentes no grafo. A extensão é automaticamente .graph. Os vértices são numerados a partir de zero e os vértices de partida são os primeiros. Ao ser executado o grafo, os vértices de partida são automaticamente colocados na fUE. Cada vértice pode ser do tipo: *Normal* (N), *Branch* (B), *Call* (CL), *Begin Recursive* (BR), *Begin Non Recursive* (BNR), *End* (E), *Reset* (R), *Consumer* (C) ou *End of Program* (EP). No caso de vértices com duas entradas, pode haver fichas de dados previamente à espera dos respectivos pares (*waitTokens*), mas é necessário especificar a qual entrada elas pertencem: *gate1* ou *gate2*.

Após o preenchimento de todos os vértices, o número de vértices de partida e quantas vezes eles são acionados serão requisitados. É possível executar um mesmo código em paralelo, quantas instâncias forem necessárias. É requisitado que o número de vértices de partida acionados seja igual ao de fichas resultado.

A.5 Arquivos

A seguir são relacionados os arquivos que constituem o pacote gMDMS. Resume-se também a organização do código por esses arquivos.

- gMDMS.c
- global.h e global.c
- program.h e program.c

- EU.h e EU.c
- MU.h e MU.c
- ISU.h e ISU.c
- TQ.h e TQ.c
- IQ.h e IQ.c
- Makefile

O módulo "global.*" contém as definições auxiliares globais a todo o sistema, bem como rotinas auxiliares, como a de erro ("Error(msg)"). O módulo "program.*" é o núcleo do sistema e contém todas as definições e rotinas básicas. Os módulos "EU.*", "MU.*" e "ISU.*" contêm as definições e rotinas específicas de cada unidade. Os módulos "TQ.*" e "IQ.*" contêm, respectivamente, as definições das filas que manipulam pacotes de fichas de dados e pacotes executáveis. E, o módulo "gMDMS.c" contém o laço global do sistema e as rotinas de entrada, de manipulação dos arquivos de resultados e de apresentação dos resultados.

Dentre os programas gerados por esta ferramenta, os mais importantes para esta dissertação são os que seguem:

- SUM.graph
- ProgramTest.graph
- SUML.graph

Seus grafos e resultados de simulação são encontrados nos capítulos 3 e 5.

Apêndice B

CPG – Guia do Usuário

A geração de programas via gMDMS exige a entrada de todos os vértices, um por um. Esta tarefa dificulta a geração de programas com grande número de instruções que não tenham chamadas recursivas ou laços. Para contornar em parte esta deficiência, foram implementados dois geradores de Programas-Constantes. Os geradores CPG1 e CPG2 (Constant-Program Generator) geram programas dos respectivos grupos 1 e 2 dos Programas-Constantes descritos no capítulo 3, seção 3.1.1.

CPG1 e CPG2 usam as mesmas estruturas de dados de gMDMS. Por isto, são necessários os módulos "global.*" e "program.*", além dos arquivos "CPG1.c" e "CPG2.c", na fase de compilação.

As interfaces de CPG1 e CPG2 são descritas a seguir:

"CPG1 programName parallelism cycles match"
"CPG2 programName parallelism cycles beginWave endWave"

- programName – Nome do programa a ser gerado. A extensão é automaticamente ".graph", por compatibilidade com gMDMS.
- parallelism – Valor do paralelismo médio do programa. Corresponde ao número de vértices paralelos organizados transversalmente pelo programa, ou seja, o Pmax do programa. Nos programas analisados anteriormente, este parâmetro foi constantemente 50.
- cycles – Número de ciclos do programa. Corresponde ao tamanho do caminho crítico do programa. Nos programas analisados anteriormente, este parâmetro foi constantemente 100.

- match – Corresponde ao tamanho da faixa *wait/match*, ou seja, ao número de vértices organizados transversalmente que têm dois argumentos de entrada e dois de saída e provocam operações *wait/match*. Este parâmetro pode variar de 0 a "parallelism".

- beginWave – Corresponde ao número do ciclo onde a faixa transversal de *wait/match* começa. Para que a faixa comece junto com o programa, deve-se entrar com o valor 1.

- endWave – Corresponde ao número do ciclo onde a faixa transversal de *wait/match* termina. Para que a faixa termine junto com o programa, deve-se entrar "cycles".

Bibliografia

[AA82] Tilak Agerwala and Arvind. Data Flow Systems. *IEEE Software*, Vol. 15, No. 2, pp. 10–13, February 1982.

[ACD74] Thomas L. Adam, K. M. Chandy, and J. R. Dickson. A Comparison of List Schedules for Parallel Processing Systems. *Communications of ACM*, Vol. 17, No. 12, pp. 685–690, December 1974.

[Ack82] William B. Ackerman. Data Flow Languages. *IEEE Software*, Vol. 15, No. 2, pp. 15–25, February 1982.

[AI87] Arvind and R. A. Iannucci. Two Fundamentals Issues in Multiprocessing. In S. S. Thakkar, editor, *Selected Reprints on Dataflow and Reduction Architectures*, pp. 140–164, 1987.

[Ash85] E. A. Ashcroft. Eazyflow Architecture. Technical Report CLS-147, Computer Science Laboratory SRI International, April 1985.

[Bab84] R. G. Babb. Parallel Processing with Large-Grain Data Flow Techniques. *Computer*, pp. 55–61, July 1984.

[Bac78] John Backus. Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs. *Communications of ACM*, Vol. 21, No. 8, pp. 613–641, August 1978.

[BE87] Richard Buehrer and Kattamuri Ekanadham. Incorporating Data Flow Ideas into von Neumann Processors for Parallel Execution. *IEEE Transactions on Computers*, Vol. C-36, No. 12, pp. 1515–1522, December 1987.

[BFHS88] J. Blazewicz, G. Finke, R. Haupt, and G. Schmidt. New Trends in Machine Scheduling. *European Journal of Operational Research*, Vol. 37, No. 3, pp. 303–317, December 1988. North-Holland.

[BG90a] Micah Beck and Jean-Luc Gaudiot. Static Schedulig for Dynamic Dataflow Machines. *Journal of Parallel and Distributed Computing*, Vol. 10, No. 4, pp. 279–288, December 1990.

[BG90b] A. P. Wim Böhm and John R. Gurd. Iterative Instructions in the Manchester Dataflow Computer. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, pp. 129–139, April 1990.

[BI92] B. Bonchev and M. Iliev. A Hybrid Dataflow Architecture with Multiple Tokens. *Lect. Notes Comput. Sci.*, Vol. 634, pp. 737–742, 1992.

[Bok81] Shahid H. Bokhari. On the Mapping Problem. *IEEE Transactions on Computers*, Vol. C-30, No. 3, pp. 207–214, March 1981.

[Cat81] Arthur João Catto. *Nondeterministic Programming in a Dataflow Environment.* PhD thesis, University of Manchester, June 1981.

[CC93] Benedito Aparecido Cruz and Arthur J. Catto. Uma Proposta para Simplificação do Emparelhamento da Dados em Máquinas de Fluxo de Dados. In *V Simpósio Brasileiro de Arquiteturas de Computadores – Processamento de Alto Desempenho – Anais*, Outubro 1993.

[Cha71] Donald D. Chamberlin. The "Single-assignment" approach to parallel processing. In *Fall Joint Computer Conference*, pp. 263–269, 1971.

[CK88] Thomas Casavant and Jon G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Transactions on Software Engineering*, Vol. 14, No. 2, pp. 141–154, February 1988.

[Cof75] E. G. Coffman Jr. Selecting a Scheduling Rule that Meets Pre-Specified Response Time Demands. In *Proc. Fifth Symposium of Operating System Principles*, 1975. published as Operating Systems Review, 9, 5, ACM, NY, pp. 187-191.

[Cof76] E. G. Coffman Jr. *Computer and Job–Shop Scheduling Theory.* Wiley–Interscience, New York, 1976.

[CR75] K. M. Chandy and P. F. Reynolds. Scheduling Partially Ordered Tasks with Probabilistic Execution Times. In *Proc. Fifth Symposium of Operating System Principles*, 1975. published as Operating Systems Review, 9, 5, ACM, NY, pp. 169-177.

[Cru] Benedito Aparecido Cruz. Uma Proposta para Simplificação do Emparelhamento da Dados em Máquinas de Fluxo de Dados. Tese de Mestrado em elaboração. Departamento de Ciência da Computação, UNICAMP.

[CS90] T. C. E. Cheng and C. C. S. Sin. A State-of-the-Art Review of Parallel-Machine scheduling Research. *European Journal of Operational Research*, Vol. 47, pp. 271–292, 1990. North-Holland.

[Das89] Subrata Dasgupta. *Computer Architecture: a modern synthesis*. Wiley, 1989.

[Den85] Jack B. Dennis. Models of Data Flow Computation. In M. Brov, editor, *Control Flow and Data Flow: Concepts of Distributed Programming*, pp. 345–398. Springer-Verlag Berlin Heidelberg, 1985.

[DK82] Alan L. Davis and Robert M. Keller. Data Flow Program Graphs. *IEEE Software*, Vol. 15, No. 2, pp. 26–41, February 1982.

[DM80] Prabuddha De and Thomas E. Morton. Scheduling to Minimize Makespan on Unequal Parallel Processor. *Decision Science*, Vol. 11, pp. 586–602, 1980.

[ERL90] Hesham El-Rewini and T. G. Lewis. Scheduling Parallel Program Tasks onto Arbitrary Target Machines. *Journal of Parallel and Distributed Computing*, Vol. 9, pp. 138–153, 1990.

[ERL92] Hesham El-Rewini and Ted Lewis. Scheduling Parallel Program Tasks onto Arbitrary Target Machines. Parallel research combined reports, Oregan State University Computer Science Department, 1992. Edited by Ted Lewis.

[FB73] Eduardo B. Fernández and Bertram Bussell. Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules. *IEEE Transactions on Computers*, Vol. C-22, No. 8, pp. 745–751, August 1973.

[FL75] Eduardo B. Fernández and T. Lang. Computation of Lower Bounds for Multiprocessor Schedules. *IBM Journal Of Research and Development*, Vol. 19, No. 5, pp. 435–444, September 1975.

[Gar92] Claudio Garcia. *Escalação Estática Sub-Ótima de Tarefas Parcialmente Ordenadas em Redes de Transputers.* PhD thesis, Escola Politécnica da Universidade de São Paulo, 1992.

[GB87] D. Ghoshal and L. N. Bhuyan. Analytical Modeling and Architectural Modifications of a Dataflow Computer. In *Proceedings of* 14^{th} *International Symposium on Computer Architecture*, pp. 81–89, Los Alamitos, Calif., 1987. IEEE Computer Society Press.

[GG73] M. R. Garey and R. L. Graham. Bounds on Scheduling with Limited Resources. In *Proc. Fourth Symposium of Operating System Principles*, 1973. published as Operating Systems Review, 7, 4, ACM, NY, pp. 104-111.

[GJ79] Michael R. Garey and David S. Johnson. *Computer and Intractability; a Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[GKW85] J. R. Gurd, C. C. Kirkham, and I. Watson. The Manchester Prototype Dataflow Computer. *Communications of ACM*, Vol. 28, No. 1, pp. 34–52, January 1985.

[Gon77] Mario J. Gonzalez Jr. Deterministic Processor Scheduling. *Computing Surveys*, Vol. 9, No. 3, pp. 173–204, September 1977.

[GPK82] D. D. Gajski, D. A. Padua, and D. J. Kuck. A Second Opinion on Data Flow Machines and Languages. *IEEE Software*, Vol. 15, No. 2, pp. 58–68, February 1982.

[Gra72] R. L. Graham. Bounds on Multiprocessing Anomalies and Related Packing Algorithms. *AFIPS Conf. Proc.*, Vol. 40, pp. 205–217, 1972.

[GS92] John R. Gurd and D. F. Snelling. Manchester Data-Flow: A Progress Report. In 6^{th} *ACM International Conference on Supercomputing*, pp. 216–225, July 1992.

[Gur85] J. R. Gurd. The Manchester Dataflow Machine. *Computer Physics Communications*, Vol. 37, No. 1, pp. 49–62, July 1985.

[GW83] John Gurd and Ian Watson. Preliminary Evaluation of a Prototype Dataflow Computer. *IFIP*, pp. 545–551, 1983.

[HCAA93] J. Hicks, D. Chiou, B. S. Ang, and Arving. Performance Studies of Id on the Monsoon Dataflow System. *J. Parallel Distrib. Comput.*, Vol. 18, No. 3, pp. 273–300, 1993.

[HCAL89] Jing-Jang Hwang, Yuan-Chieh Chow, Frank D. Anger, and Chung-Yee Lee. Scheduling Precedence graphs in Systems with Interprocessor Communication Times. *SIAM Journal on Computing*, Vol. 18, No. 2, pp. 244–257, April 1989.

[Her75] U. Herzog. Optimal Scheduling Strategies for Real-Time Computers. *IBM Journal Of Research and Development*, Vol. 19, No. 5, pp. 494–504, September 1975.

[Hu61] T. C. Hu. Parallel Sequencing and Assembly Line Problems. *Operations Research*, Vol. 9, pp. 841–848, November 1961.

[Kam94] Carlos Alberto Kamienski. Armazenamento de Resultados em uma Arquitetura de Fluxo de Dados, Março 1994. Tese de mestrado em Ciência da Computação, Universidade Estadual de Campinas, Campinas, São Paulo.

[KBB87] Krishna M. Kavi, Billy P. Buckles, and U. Narayan Bhat. Isomorphisms Between Petri Nets and Dataflow Graphs. *IEEE Transaction on Software Engineering*, Vol. SE-13, No. 10, pp. 1127–1134, October 1987.

[Kel85] Robert M. Keller. Rediflow Architecture Prospectus. Technical Report UUCS-85-105, Department of Computer Science, University of Utah, Salt Lake City, UT 84112, August 1985.

[Kim87] Yeong-Dae Kim. On the Superioty of a Backward Approach in List Scheduling Algorithms for Multi-Machine Makespan Ploblems. *International Journal of Production Research*, Vol. 25, No. 12, pp. 1751–1759, December 1987.

[Kir90] Chris Kirkham. The Manchester Dataflow Project. In Terry J. Fountain and Malcolm J. Shute, editors, *Multiprocessor Computer Architectures*, chapter 7. Elsevier Science, 1990.

[KL88] Boontee Kruatrachue and Ted Lewis. Grain Size Determination for Parallel Processing. *IEEE Software*, Vol. 5, No. 1, pp. 23–32, January 1988.

[KL92a] Boontee Kruatrachue and Ted Lewis. Duplication Schedulinh Heuristic. Parallel research combined reports, Oregan State University Computer Science Department, 1992. Edited by Ted Lewis.

[KL92b] Boontee Kruatrachue and Ted Lewis. Optimal Grain Determination. Parallel research combined reports, Oregan State University Computer Science Department, 1992. Edited by Ted Lewis.

[KN84] Hironori Kasahara and Seinosuke Narita. Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing. *IEEE Transaction on Computers*, Vol. c-33, No. 11, pp. 1023–1029, November 1984.

[Koh75] Walter H. Kohler. A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems. *IEEE Transaction on Computers*, Vol. c-24, No. 12, pp. 1235–1238, December 1975.

[KSY92] Y. Kodama, S. Sakai, and Y. Yamaguchi. A Prototype of a Highly Parallel Dataflow Machine EM-4 and Its Preliminary Evaluation. *Future Gener. Comput. Syst.*, Vol. 7, No. 2/3, pp. 199–209, 1992.

[LC92] Paulo A. R. Lorenzo and Arthur J. Catto. Escalonamento de Processos Considerando Atrasos de Comunicação. In *IV Simpósio Brasileiro de Arquiteturas de Computadores – Processamento de Alto Desempenho – Anais*, pp. 537–545, Outubro 1992. Processing Scheduling with Communication Delays. In Portuguese.

[LER92] Ted G. Lewis and Hesham El-Rewini. *Introduction to Parallel Computing*. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[LGC93] Paulo Lorenzo, Munehiro Goto, and Arthur J. Catto. The Impact of the Instruction Scheduling in the MDFM. In *Tokai-Section Joint Convention Record of the Six Institutes of Electrical and Related Engineers*, pp. 377, Nagano, Japan, October 1993.

[LGC94] Paulo Lorenzo, Munehiro Goto, and Arthur J. Catto. Sequential Blocking on the MDFM. In *Tokai-Section Joint Convention Record of the Six Institutes of Electrical and Related Engineers*, pp. 398, Gifu, Japan, October 1994.

[LHCA88] Chung-Yee Lee, Jing-Jang Hwang, Yuan-Chieh Chow, and Frank D. Anger. Multiprocessor Scheduling with Interprocessor Communication Delays. *Operations Reseach Letters*, Vol. 7, No. 3, pp. 141–147, June 1988.

[LS75] Shui Lam and Ravi Sethi. Analysis of Level Algorithm for Preemptive Scheduling. In *Proc. Fifth Symposium of Operating System Principles*, 1975. published as Operating Systems Review, 9, 5, ACM, NY, pp. 178–186.

[LY89] Joseph Y-T. Leung and Gilbert H. Young. Minimizing Schedule Length Subject to Minimum Flow Time. *SIAM Journal on Computing*, Vol. 18, No. 2, pp. 314–326, April 1989.

[Mag80] Gyula A. Magó. A Cellular Computer Architecture for Functional Programing. In *COMPCON Spring 80-VLSI: New Architectural Horizons*, pp. 179–187, 1980.

[MC69] Richard R. Muntz and Edward G. Coffman, Jr. Optimal Preemptive Scheduling on Two-Processor Systems. *IEEE Transaction on Computers*, Vol. c-18, No. 11, pp. 1014–1020, November 1969.

[NA89] Rishiyur S. Nikhil and Arvind. Can Dataflow Subsume von Neumann Computing? In *Proceedings of the* 16^{th} *Annual International Symposium on Computer Architectures*, pp. 262–273, May 1989.

[Nav90] Philippe O. A. Navaux. *Processamento Pipeline e Processamento Vetorial.* VII Escola de Computação, São Paulo, IME-USP, 1990.

[NBM93] W. A. Najjar, A. P W. Boehm, and W. M. Miller. A Quantitative Analysis of Dataflow Program Execution - Prelimiraries to a Hybrid Design. *J. Parallel Distrib. Comput.*, Vol. 18, No. 3, pp. 314–326, 1993.

[OKSY92] K. Okamoto, Y. Kodama, S. Sakai, and Y. Yamaguchi. Methodoligies in Development and Testing of the Dataflow Machine EM-4. *Parallel Comput.*, Vol. 18, No. 8, pp. 901–912, 1992.

[Pap91] Gregory M. Papadopoulos. *Implementation of a General-Purpose Dataflow Multiprocessor.* The MIT Press, Cambridge, Massachusetts, 1991.

[PI77] S. S. Panwalkar and Wafik Iskander. A Survey of Scheduling Rules. *Operations Research*, Vol. 25, No. 1, pp. 45–61, January-February 1977.

[PK87] Constantine D. Polychronopoulos and David J. Kuck. Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers. *IEEE Transaction on Computers*, Vol. c-36, No. 12, pp. 1425–1439, December 1987.

[RCG72] C. V. Ramamoorthy, K. M. Chandy, and Mario J. Gonzalez, Jr. Optimal Scheduling Strategies in a Multiprocessor System. *IEEE Transactions on Computers*, Vol. C-21, No. 2, pp. 137–146, February 1972.

[RS87] Carlos A. Ruggiero and John Sargeant. Control of Parallelism in the Manchester Dataflow Computer. In *Lecture Notes in Computer Science*, volume 274, pp. 1–15. Springer-Verlag, 1987.

[Sar89] Vivek Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. The MIT Press, 1989.

[SC92] Sérgio R. P. da Silva and Arthur J. Catto. Modelagem e Análise de Desempenho de uma Arquitetura de Fluxo de Dados. In *IV Simpósio Brasileiro de Arquiteturas de Computadores – Processamento de Alto Desempenho – Anais*, pp. 159–174, Outubro 1992.

[Sow87] M. Sowa. A Method for Speeding up Serial Processing in Dataflow Computers by Means of a Program Counter. *The Computer Journal*, Vol. 30, No. 4, pp. 289–294, 1987.

[Sto77] Harold S. Stone. Multiprocessor Scheduling with the Aid of Network Flow Algorithms. *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, pp. 85–93, January 1977.

[TBH82] Philip C. Treleaven, David R. Brownbridge, and Richard P. Hopkins. Data-Driven and Demand-Driven Computer Architecture. *Computing Surveys*, Vol. 14, No. 1, pp. 93–143, March 1982.

[Tow86] Don Towsley. Allocating Programs Containing Branches Loops Within a Multiple Processor System. *IEEE Transaction on Software Engineering*, Vol. 12, No. 10, pp. 1018–1024, October 1986.

[Ull75] J. D. Ullman. NP-Complete Scheduling Problems. *Journal of Computer and System Sciences*, Vol. 10, pp. 384–393, 1975.

[VBJ90] Harrick M. Vin, Francine Berman, and James S. Mattson Jr. Efficient Data-Driven Evaluation: Theory and Implementation. *Journal of Parallel and Distributed Computing*, Vol. 10, No. 4, pp. 367–385, December 1990.

[VC92] Marcos C. Visoli and Arthur J. Catto. Tratamento de Código Seqüencial no Modelo de Fluxo de Dados (Treatment of Sequential Code in Data Flow Model). In *IV Simpósio Brasileiro de Arquiteturas de Computadores – Processamento de Alto Desempenho – Anais*, pp. 147–158, October 1992. In Portuguese.

[Vee86] Arthur H. Veen. Dataflow Machine Architecture. *ACM Computing Surveys*, Vol. 18, No. 4, pp. 365–396, December 1986.

[Vis] Marcos Cezar Visoli. Tratamento de Código Seqüencial no Modelo de Fluxo de Dados. Tese de Mestrado em elaboração. Departamento de Ciência da Computação, UNICAMP.

[WG82] Ian Watson and John Gurd. A Pratical Data Flow Computer. *IEEE Software*, Vol. 15, No. 2, pp. 51–57, February 1982.

[Wil93] Gregory V. Wilson. A Glossary of Parallel Computing Terminology. *IEEE Parallel & Distributed Technology*, pp. 52–67, February 1993.

[YSK91] Yoshinori Yamaguchi, Shuichi Sakai, and Yuetsu Kodama. Synchonization Mechanisms of a Highly Parallel Dataflow Machine EM-4. *IEICE Transactions*, Vol. E74, No. 1, pp. 204–213, January 1991.

[YSY90] Toshitsugu Yuba, Toshio Shimada, and Yoshinori Yamaguchi. Dataflow Computer Development in Japan. In *Proceedings of the 1990 International Conference on Supercomputing*, pp. 140–147, September 1990.