ESCALONAMENTO DE PROCESSOS CONSIDERANDO ATRASOS DE COMUNICAÇÃO

Paulo A. R. Lorenzo[1]
Departamento de Ciência da Computação
UNICAMP
Caixa Postal 6065
13081-970 — Campinas — SP
(0192) 39-8442
lorenzo@dcc.unicamp.br

Arthur J. Catto[2]
Departamento de Ciência da Computação
UNICAMP
Caixa Postal 6065
13081-970 — Campinas — SP
(0192) 39-8442
catto@dcc.unicamp.br

RESUMO

Um dos problemas críticos em multiprocessadores é o escalonamento eficiente de processos, que engloba a partição do programa em módulos e seu endereçamento às unidades de processamento.

Problemas como a determinação do tamanho ideal dos módulos e a análise do custo de comunicação ainda não tiveram soluções satisfatórias. Técnicas genéricas não conseguiram grande resultado, devido à correlação entre as questões pontos e as arquiteturas alvo. Parâmetros como número de unidades de processamento e características da rede de intercomunicação têm que ser considerados em qualquer proposta de escalonamento.

Iniciar os processos o mais cedo possível não garante um bom escalonamento. Esse procedimento, ao contrário do esperado, pode prejudicar o desempenho do sistema, devido à não consideração dos atrasos de comunicação existentes.

Este artigo aborda fatores importantes para um escalonamento eficiente. Destacam-se a importância e a necessidade de um tratamento adequado dos atrasos de comunicação envolvidos no processamento paralelo.

ABSTRACT

One of the critical problems in multiprocessors is the efficient scheduling of tasks, which comprises program partitioning into modules — called grains — and their allocation to the processing units.

Problems such as ideal grain size determination and communication delay analysis have no satisfying solution yet. Generic techniques do not give good results, because of the dependencies between those problems and the target architectures. Parameters such as the number of processors and intercommunication network caracteristics must be taken into account in any scheduling proposal.

Initiating processes as early as possible does not guarantee proper scheduling. Instead, such procedure can hinder performance, if existing communication delays are not considered.

This paper addresses important issues for efficient scheduling. The importance and need for an adequate treatment of the communication delays associated to parallel processing are highlighted.

[1] Mestrando em Ciência da Computação.

[2] Professor Assistente Doutor, Ph.D. in Computer Science, University of Manchester, 1981.

1 INTRODUÇÃO

O escalonamento de processos em um ambiente de computação paralela é um dos pontos fundamentais para a eficiência do sistema. O escalonamento depende das características da linguagem de programação, do modelo de computação e dos parâmetros utilizados pelo escalonador.

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

Este artigo descreve o estado atual de uma pesquisa na área de escalonamento de processos [Lor], levando em conta a influência dos atrasos de comunicação, desenvolvida no âmbito do projeto Wolf, uma iniciativa de pesquisadores da Unicamp e do Instituto de Física e Química de São Carlos da Universidade de São Paulo para o desenvolvimento de arquiteturas de fluxo de dados.

A seguir, são descritos problemas que ainda permanecem sem o devido tratamento em ambientes paralelos. A partir deles, apresentam-se linguagens paralelas e os modelos von Neumann e de fluxo de dados. Sobre este último, é feita uma análise do conceito de *eager evaluation*. Nas duas últimas seções discutem-se a importância de se considerar os atrasos de comunicação e apresentam-se conclusões e trabalhos futuros.

2 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 o custo comunicação [ERL90, KL88].

A partição do programa em módulos é necessária para garantir que a granuralidade 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 m tarefas para n UPs é NP-completo [Ull75, Cof76, GJ79]. Com o objetivo de se alcançar o melhor desempelho possível, várias pesquisas recentes têm tratado deste tema [ERL90, KL88, Sar89, YSK91, VBJ90, CYLA88, Tow86].

Qual o melhor tamanho de cada grão, como dividir o programa em 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 políticas 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].

3 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 pela qual os programadores normalmente pensariam para resolver um problema [Ack82]. Devido à origem seqüencial das linguagens convencionais, vários cuidados e técnicas têm que ser adotados na 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 não naturais [Cha71, Ack82, Das89].

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

4 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]. O fluxo de controle, por isso, é implicitamente seqüencial, mesmo sendo tolerável o uso explícito de operadores de controle que proporcionam paralelismo [TBH82].

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 processadores von Neumann, paralelizando módulos de código, acarreta uma sobrecarga de sincronização, devido ao controle de consistência dos dados compartilhados. Para amenizar esse custo, deve ser otimizada a escolha dos grãos. Embora conseguida satisfatoriamente no caso de acesso regular a estruturas de dados seqüenciais, nos casos mais gerais a solução desse problema permanece uma tarefa difícil [AA82, Das89].

As pesquisas têm privilegiado duas vertentes na solução dos problemas de tamanho do grão e de seu escalonamento [KL88, Sar89]: listas de escalonamento (p.ex., *load balancing*) [Gra72] e *large-grain dataflow* [Bab84]. Contudo, a análise de custo de comunicação entre processadores 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 dependentes, com alto custo de comunicação, poderiam ser mais eficientemente escalonados para o mesmo processador, do que para processadores diferentes [KL88].

5 MODELO DE FLUXO DE DADOS

Nos modelos 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, o fluxo dos dados e o de controle são idênticos [TBH82] e o sincronismo entre instruções é garantido implicita e minimamente pelo percurso dos dados [Das89].

5.1 EAGER EVALUATION × RECURSOS LIMITADOS

Programas no modelo de fluxo de dados são usualmente descritos como grafos orientados, chamados grafos de fluxo de dados (GFD). Num GFD, cada vértice representa uma instrução e cada aresta a dependência de dados entre duas instruções. A Figura 1 apresenta um trecho de programa

e sua representação em GFD [TBH82, DK82]. Essa representação destaca um paralelismo de granularidade fina, ao nível de instrução [Das89].

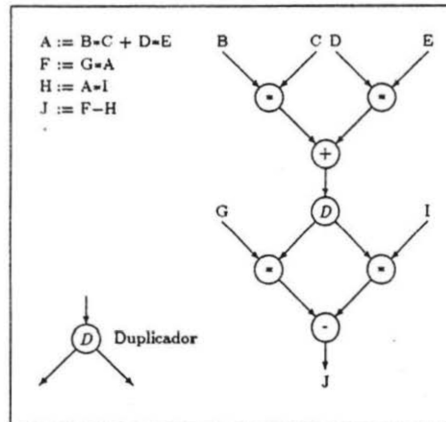

Figura 1: Um trecho de programa e seu Grafo de Fluxo de Dados

O modelo de fluxo de dados puro adota o princípio de *eager evaluation*: as instruções ficam prontas para a execução assim que todos os seus argumentos estejam disponíveis e todas as instruções prontas para a execução são processadas [VBJ90, TBH82]. Entretanto, não é considerado o fato de que os recursos de processamento são limitados e que instruções cujos resultados não serão imediatamente aproveitados podem ocupar o lugar de instruções do caminho crítico [VBJ90]. Isso pode afetar o desempenho da máquina.

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

5.2 PARTIÇÃO

GFDs ressaltam, também, uma intensa comunicação entre instruções, o que torna a rede de comunicação o gargalo do modelo de fluxo de dados [YSY90]. Por mais paralelismo que exista no ambiente, a intensidade do fluxo de dados entre instruções, que se reflete numa freqüente comunicação entre processadores, é um fator limitante no desempenho do sistema.

No caso de trechos seqüenciais, a paralelização só aumenta o tempo de execução, devido ao custo desnecessário de comunicação. O empacotamento desses trechos num só grão 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 de custo de comunicação entre instruções, parâmetro este vinculado à arquitetura em questão. Nessa abordagem, os grãos de tamanhos variados evitam a paralelização prejudicial.

5.3 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 [CK88].

O escalonamento estático é livre do custo de determinação em tempo de execução, mas é inadequado ao tratamento de laços e desvios [ERL90]. Já o escalonamento dinâmico não aproveita técnicas mais eleboradas que necessitam de grande processamento ou retrocessos na ordem dos

módulos. A união desses estilos permite um tratamento adequado de laços e desvios [BG90, Tow86] e o uso de métodos mais eficientes.

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

O elemento que seleciona o processador 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 políticas mais eleboradas de escalonamento pode representar um grande ganho para o sistema [YSK91, VBJ90, KL88, ERL90].

Por exemplo, considere-se a Figura 2a, que mostra um GFD representando um trecho de programa. Supondo que o tempo de execução de todos os nodos seja o mesmo e que não haja atrasos de comunicação, a execução desse trecho, numa arquitetura com dois processadores, segundo o modelo de fluxo de dados puro, poderia ser representada pelo *Gantt chart* da Figura 2b. Pode-se observar facilmente que um escalonamento mais rigoroso pode produzir um melhor resultado, como, por exemplo, o da Figura 2c.

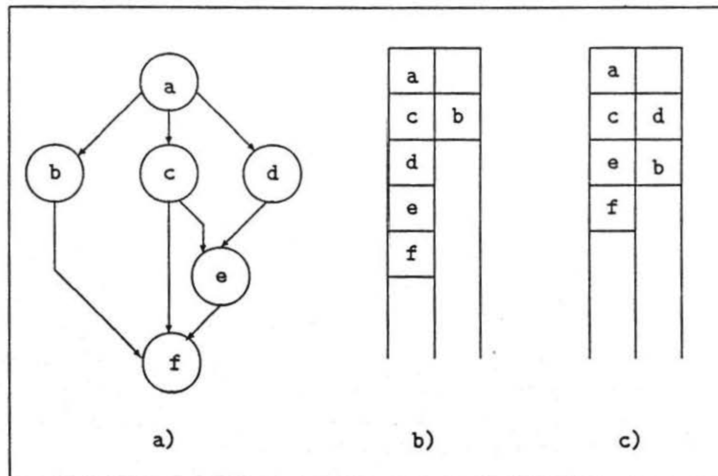

Figura 2: Impacto de uma política de escalonamento: a) GFD de um trecho de um programa, b) *Gantt chart* do princípio *eager evaluation*, c) *Gantt chart* de uma política de escalonamento.

6 ESCALONAMENTO CONSIDENRANDO ATRASOS DE COMUNICAÇÃO

Na busca de algoritmos capazes de gerar um escalonamento mais próximo do ótimo, é indispensável a análise dos custos de comunicação do sistema. Os atrasos gerados pelo envio dos resultados de um módulo para outros dependentes, mas executados em UPs distintas, podem fazer com que programas paralelos se comportem de modo inesperado.

Entretanto, a maioria das pesquisas já realizadas sobre escalonamento de processos em ambientes paralelos não considera atrasos de comunicação. Para elas, paralelizar é o grande meio de se obter um menor tempo de execução.

Na prática, o que se observa é que nem sempre o escalomento com maior paralelismo é o mais eficiente. Distribuir módulos paralelizáveis para quantas UPs forem possíveis tende a aumentar os atrasos de comunicação, o que contribui para o aumento do tempo de execução [KL92a]. A Figura 3a reproduz o GFD da Figura 2a, agora considerando que só não existem atrasos na comunicação entre módulos escalonados para uma mesma UP — como se eles estivessem empacotados num único módulo — e que os atrasos existentes são de duas unidades de tempo. O escalonamento do *Gantt chart* da Figura 3b busca um alto desempenho através da máxima paralelização. Devido aos atrasos de comunicação, em alguns casos é mais eficiente serializar a execução dos módulos, como no *Gantt chart* da Figura 3c.

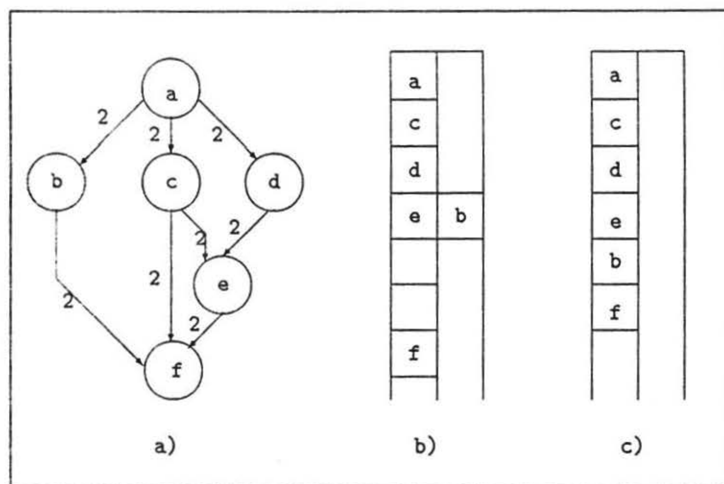

Figura 3: Impacto de uma política de escalonamento com análise de atrasos de comunicação: a) GFD de um trecho de um programa com atrasos de comunicação iguais a duas unidades de tempo, b) *Gantt chart* de um escalonamento que enfatiza a paralelização, c) *Gantt chart* de uma política de escalonamento com análise dos custos de comunicação.

Algumas pesquisas recentes têm abordado políticas de escalonamento que consideram atrasos de comunicação [KL88, ERL90, CYLA88]. Destas, pode-se destacar a heurística *Duplication Scheduling Heuristic* (*DSH*) [KL88], que usa o conceito de duplicação de tarefas. DSH é a primeira abordagem desse conceito no escalonamento de processos.

7 CONCLUSÃO E TRABALHOS FUTUROS

Como exposto, os modelos de computação paralela não oferecem tratamento adequado para a partição de um programa em módulos e para o escalonamento destes entre as UPs . Não há, nesse aspecto, nenhum desenvolvimento satisfatório, que considere fatores da arquitetura usada, tais como: número de UPs e atrasos de comunicação entre elas.

As implementações do modelo de fluxo de dados consideram que o número de UPs é tão grande que o supõem infinito e confiam no balaceamento ótimo e natural dos módulos na execução. Na prática, ocorre que o número de UPs é finito e, por isso, é necessário gerenciar o seu uso.

As políticas de escalonamento têm que considerar também parâmetros como atrasos de comunicação e número de UPs, não somente o máximo de paralelismo possível. Uma importante propriedade de um 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 causar um grande atraso, o escalonador deve ter a capacidade de decisão de não usar as UPs adicionais [KL92a].

Dentro do projeto Wolf estão sendo desenvolvidos dois trabalhos de tese que abordam esse assunto [Lor, Vis]. Em [Vis], estão sendo desenvolvidas técnicas de empacotamento de módulos. Em [Lor], está sendo feita uma análise dos principais parâmetros de um escalonador eficiente, como atrasos de comunicação, para a identificação de políticas de escalonamento adequadas a sistemas paralelos. Como esse é um problema que atinge os modelos von Neumann, de fluxo de dados e de demanda, os resultados devem ser híbridos e com aspectos genéricos.

As experiências com arquiteturas de fluxo de dados demonstram que mesmo com a utilização de um modelo de computação originalmente paralelo, é necessária a utilização de políticas de escalonamento mais elaboradas. Pelo fato de o modelo von Neumann ser o mais rico em pesquisas e o modelo de demanda ser de origem paralela, a análise dos esforços desenvolvidos em ambos é essencial ao desenvolvimento deste trabalho.

Nesse estudo, estão sendo destacados aspectos tradicionalmente não considerados, mas que têm uma relevante influência na qualidade do escalonador. Com base nesse levantamento e nas conclusões de [Vis], serão selecionadas políticas de escalonamento e critérios de qualidade que elas devem satisfazer.

Para o modelo de fluxo de dados, serão realizadas simulações desse grupo de políticas. A análise comparativa desses resultados oferecerá suporte para propostas visando melhorar o desempenho de arquiteturas de fluxo de dados.

REFERÊNCIAS

[AA82] Tilak Agerwala and Arvind. Data Flow Systems. *IEEE Software*, 15(2):10–13, February 1982.

[Ack82] William B. Ackerman. Data Flow Languages. *IEEE Software*, 15(2):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*, pages 140–164, 1987.

[Bab84] R. G. Babb. Parallel Processing with Large-Grain Data Flow Techniques. *Computer*, pages 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*, 21(8):613–641, August 1978.

[BG90] Micah Beck and Jean-Luc Gaudiot. Static Schedulig for Dynamic Dataflow Machines. *Journal of Parallel and Distributed Computing*, 10(4):279–288, December 1990.

[Cha71] Donald D. Chamberlin. The "Single-assignment" approach to parallel processing. In *Fall Joint Computer Conference*, pages 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*, 14(2):141–154, February 1988.

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

[CYLA88] Yuan-Chieh Chow Chung-Yee Lee, Jing-Jang Hwang and Frank D. Anger. Multiprocessor Scheduling with Interprocessor Communication Delays. *Operations Reseach Letters*, 7(3):141–147, June 1988.

[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*, pages 345–398. Springer-Verlag Berlin Heidelberg, 1985.

[DK82] Alan L. Davis and Robert M. Keller. Data Flow Program Graphs. *IEEE Software*, 15(2):26–41, February 1982.

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

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

[GPK82] D. D. Gajski, D. A. Padua, and D. J. Kuck. A Second Opinion on Data Flow Machines and Languages. *IEEE Software*, 15(2):58–68, February 1982.

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

[KL88] Boontee Kruatrachue and Ted Lewis. Grain Size Determination for Parallel Processing. *IEEE Software*, 5(1):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.

[Lor] Paulo A. R. Lorenzo. Escalonamento de Processos em uma Arquitetura de Fluxo de Dados. Tese de Mestrado em elaboração. Departamento de Ciência da Computação, UNICAMP.

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

[TBH82] Philip C. Treleaven, David R. Brownbridge, and Richard P. Hopkins. Data-Driven and Demand-Driven Computer Architecture. *Computing Surveys*, 14(1):93–143, March 1982.

[Tow86] Don Towsley. Allocating Programs Containing Branches Loops Within a Multiple Processor System. *IEEE Transaction on Software Engineering*, 12(10):1018–1024, October 1986.

[Ull75] J. D. Ullman. NP-Complete Scheduling Problems. *Journal of Computer and System Sciences*, 10:384–393, 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*, 10(4):367–385, December 1990.

[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.

[YSK91] Yoshinori Yamaguchi, Shuichi Sakai, and Yuetsu Kodama. Synchonization Mechanisms of a Highly Parallel Dataflow Machine EM-4. *IEICE Transactions*, 74(1):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*, pages 140–147, September 1990.