Modelos Computacionais de Fluxo de Dados

L. E. Buzato
C. M. F. R. Calsavara
A. J. Catto

Departamento de Ciência da Computação
IMECC - UNICAMP
Caixa Postal: 6065
13081 - Campinas/SP

Sumário

Modelos de Fluxo de Dados têm sido exaustivamente estudados como uma alternativa para a implementação de máquinas paralelas. Este trabalho descreve os principais modelos: estático e dinâmico, discutindo suas características. Alguns problemas apresentados pelos modelos e implementações realizadas são avaliados.

Abstract

Data Flow Models have been widely studied as promising candidates to parallel machine implementation. This paper describes the static and tagged models addressing their main characteristics. Related modelling and implementation problems are also discussed.

1 Introdução

Por mais de quarenta anos, a organização de computadores baseada no modelo de von Neumann manteve-se inalterada, determinando totalmente o projeto de computadores e do "software" relacionado, como por exemplo, linguagens de programação e sistemas operacionais.

Nos últimos anos, no entanto, tem-se observado um interesse crescente entre os pesquisadores por modelos de computação paralela, justificado pela impossibilidade física e tecnológica do aumento indefinido da capacidade computacional das máquinas convencionais.

A solução apontada é o desenvolvimento de modelos computacionais capazes de oferecer maior poder de processamento a partir do assincronismo e do controle distribuído da computação, divergindo assim do modelo de von Neumann cujas principais características residem no controle seqüencial da computação e na organização linear da memória. Este modelo associa ao operando uma localização específica de memória, obrigando o programador a preocupar-se não somente com o dado em si, mas também com o seu endereço. Todos os acessos a dados e instruções são realizados através do barramento de comunicação entre o processador e a memória, o que causa o aparecimento do "gargalo de von Neumann", uma grave limitação da arquitetura da máquina [1]. Os novos modelos tratam apenas os operandos como objetos da computação, não importando seu local de armazenamento.

Há duas classes principais de modelos computacionais paralelos:

- computação dirigida pela demanda de resultados ("demand-driven"),
- computação dirigida pela disponibilidade de dados ("data-driven").

Na classe "demand-driven", a necessidade de resultados dispara as operações que os geram. Na classe "data-driven", a disponibilidade de operandos dispara as operações que os utilizam.

As principais vantagens destes modelos são: a eliminação do "gargalo de von Neumann", o alto grau de paralelismo exibido e o assincronismo na execução de operações. Ambos têm também suas desvantagens: na classe "demand-driven" pode ocorrer uma sobrecarga dos elementos processadores ("lazy evaluation"); na classe "data-driven" resultados são obtidos tão logo haja disponibilidade de operandos e isto poderá implicar num eventual trabalho desnecessário ("eager evaluation").

Dentre os modelos da classe "data-driven" destaca-se o modelo de Fluxo de Dados, já suportado por algumas implementações, que será discutido neste trabalho.

2 Modelos de Computação Paralela

Antes de discutirmos o Modelo de Fluxo de Dados faremos uma breve introdução aos principais Modelos de Computação Paralela existentes [2], [3].

2.1 Modelo de Fluxo de Controle

O modelo de fluxo de controle é praticamente o único utilizado até hoje na construção de computadores. Suas características principais são: o controle é seqüencial, isto é, o controle é passado implicitamente de uma instrução para outra, e a memória é organizada como um vetor. No entanto, existem formas paralelas de fluxo de controle, como o mecanismo FORK-JOIN e o mecanismo de mensagens.

No mecanismo FORK-JOIN, a instrução FORK causa a duplicação do fluxo de controle ativando um outro ramo paralelo de computação. A instrução JOIN é responsável pela sincronização posterior desses fluxos, permitindo a reunificação do controle sobre o programa.

O modelo de fluxo de controle utilizando mensagens pode ser visto como uma generalização do modelo FORK-JOIN, onde há múltiplos FORKs seguidos de múltiplos JOINs.

Os fluxos de controle seqüencial e paralelo apresentam pontos em comum: os dados residem em memórias compartilhadas; o fluxo de controle é implicitamente seqüencial, mas operadores de controle explícito podem ser usados para se obter paralelismo e nitidamente os fluxos de controle e de dados são distintos.

2.2 Modelo de Redução

O modelo de redução utiliza o princípio matemático de função como forma semântica básica. Uma instrução é vista como uma aplicação de uma função sobre um determinado número de argumentos, com o resultado substituindo a chamada. Devido a esta abordagem, os programas são constituídos por conjuntos de expressões encaixadas, e executar um programa equivale a avaliar a função composta que o representa e retornar seus resultados.

Por exemplo, encontrar o resultado de $\mathbf{a} = (\mathbf{b}+\mathbf{1}) * (\mathbf{b}-\mathbf{c})$ implica resolver $(\mathbf{b}+\mathbf{1})$ e substituir seu resultado no ponto de chamada; resolver $(\mathbf{b}-\mathbf{c})$ e também fazer a substituição; finalmente substituir todas as referências a **a** pelo resultado da multiplicação das subexpressões. Uma referência a **a** em qualquer ponto do programa deve retornar o mesmo valor. Esta propriedade é denominada transparência referencial.

Há duas formas de redução:

- Redução de cadeias
- Redução de grafos

Na primeira forma, sempre que há referência a uma expressão, uma nova cópia é realizada e então reduzida; na segunda, a expressão é reduzida uma única vez e seu resultado substitui a expressão. Os resultados das subexpressões envolvidas no cálculo daquela que as contêm também são armazenados. Novas referências à mesma expressão não implicarão em novas reduções, bastando recuperar o resultado desejado e empregá-lo no cálculo da expressão atualmente avaliada [3], [4], [5].

Em resumo, as características básicas do modelo de redução são: as estruturas de programa, instruções e argumentos são funções; não há o conceito de reatribuição de valores à memória e o resultado da execução pode retornar um argumento simples ou complexo (funções). Recentemente, Watson [6] iniciou, em Manchester, a implementação de uma arquitetura híbrida (fluxo de dados + redução) com base nesse modelo.

2.3 Modelo de Fluxo de Dados

O modelo de fluxo de dados será detalhado na próxima seção. Suas características básicas são: os resultados parciais são representados por fichas de dados e passados entre as instruções , a execução de instruções consome as fichas que representam seus argumentos, isto é, esses valores não ficam mais disponíveis; não existe o conceito de memória compartilhada e os fluxos de controle são parcialmente pré-determinados pelas dependências funcionais entre as instruções [7].

3 O Modelo de Fluxo de Dados

3.1 O Modelo Básico

O modelo de fluxo de dados tem sido adotado como base semântica de novos sistemas e não apenas para melhorar o desempenho de processadores convencionais.

Uma operação de fluxo de dados é puramente funcional (no sentido matemático), não produzindo efeitos colaterais como resultado de sua execução.

A notação utilizada para representar os programas no modelo de fluxo de dados é a de grafos orientados acíclicos. Nestes grafos cada nó representa um operador e as arestas orientadas representam as dependências funcionais entre os operadores. Os resultados são transferidos de um operador a outro através de fichas de dados. Um estudo formal pode ser encontrado em [8].

Uma ficha pode ser representada da seguinte maneira:

< valor, aresta destino >

Um estado de um grafo de fluxo de dados é caracterizado pelo conjunto de fichas existentes no grafo em um dado momento. Uma seqüência de estados representa a história da execução do programa (figura 1). A execução de um operador corresponde ao disparo de um nó.

Os nós são disparados de acordo com as seguintes regras:

- um nó está habilitado para disparo se, e somente se, existir uma ficha em cada uma de suas arestas de entrada;
- qualquer conjunto de nós habilitados pode ser disparado simultaneamente para definir o próximo estado da computação;
- um nó é disparado removendo-se uma ficha-argumento de cada uma de suas arestas de entrada e produzindo-se uma ficha-resultado em cada uma de suas arestas de saída.

A necessidade de representação de expressões condicionais leva à introdução de operadores para desviar o fluxo das fichas de dados entre os ramos do grafo, por exemplo: um operador condicional e um filtro V (figura 2). O operador condicional passa a ficha da aresta de entrada para a aresta de saída selecionada pela ficha de

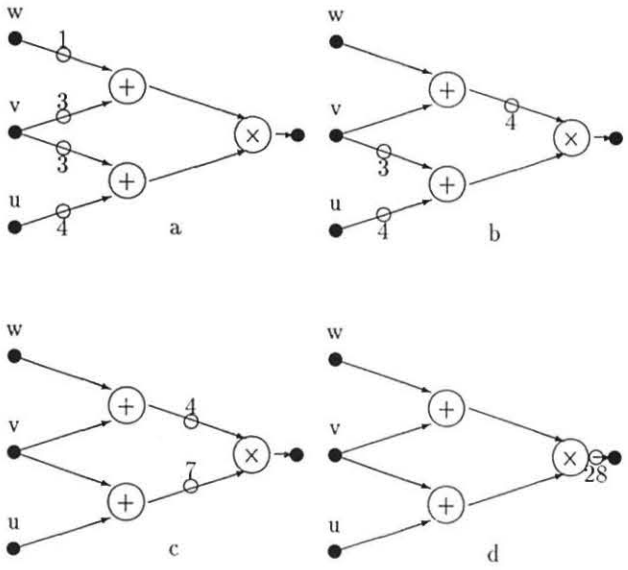

Figura 1: Possíveis estados de execução para a expressão $(u + v) * (v + w)$.

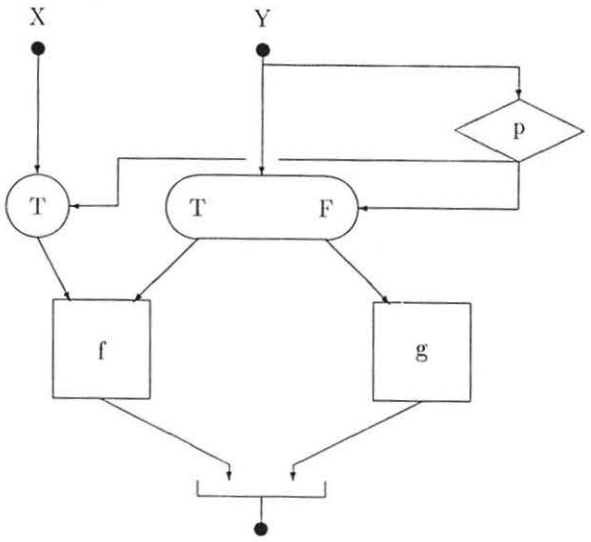

Figura 2: Operadores condicionais

controle. O filtro V passa a ficha de entrada adiante se a ficha de controle tem valor V, caso contrário, a ficha é simplesmente absorvida.

O modelo básico exige que em toda a história de um programa, não haja duas fichas com a mesma aresta destino, o que impõe a não reutilização dos grafos e implica na representação das estruturas potencialmente repetitivas utilizando sua forma linear equivalente. Nesse caso, a ativação de uma função é implementada pela substituição do grafo que a representa nos pontos de chamada. Com a incorporação do grafo da função ao grafo do programa, as substituições de argumentos para ativação da função passam a ser consideradas irrestritas, ou seja, não é necessário esperar pelo suprimento de todos os argumentos para que a ativação seja iniciada [7], [9] (figura 3).

A limitação imposta por este modelo está ligada à não reutilização de grafos, tornando-o inviável para a realização de programas de interesse prático. Também não se consegue representar recursão, já que é impossível distinguir instâncias diferentes de um mesmo grafo.

3.2 O Modelo Estático

O modelo **estático** [9] suporta uma forma restrita de reaproveitamento de grafos, permitindo que a história de um programa contenha fichas com a mesma aresta destino. No entanto, as arestas destino de todas as fichas de cada estado devem ser distintas. Como conseqüência, uma nova regra é acrescentada às anteriores:

- para que um nó esteja habilitado, não deve haver fichas em qualquer uma das suas arestas de saída.

A introdução desta regra passa a permitir encadeamento ("pipelining") [10] e o reaproveitamento do grafo para conjuntos sucessivos de dados. O encadeamento aliado ao uso de expressões condicionais traz o problema da ordenação dos resultados produzidos. Para solucioná-lo, propõe-se um operador intercalador ("merge"), que garante a preservação da ordem dos resultados e a coerência do programa executado.

As regras de disparo para o nó intercalador são:

- um nó intercalador está habilitado para disparo se, e somente se, não existir uma ficha na sua aresta de saída, uma ficha com valor lógico (assume somente os valores V ou F) estiver presente na aresta de controle e existir uma ficha na aresta de entrada selecionada pelo valor da ficha. A outra aresta de entrada pode ou não conter uma ficha.

- durante o disparo de um nó intercalador habilitado, as fichas utilizadas são removidas da aresta de controle e da aresta de entrada selecionada e uma ficha carregando o valor da ficha de entrada é produzida na saída.

A execução de expressões condicionais aliada ao encadeamento gera uma seqüência de fichas de controle contendo valores lógicos. Para que a ordenação original entre elas seja preservada, sem prejuízo do avanço da história do programa, estas fichas devem ser armazenadas

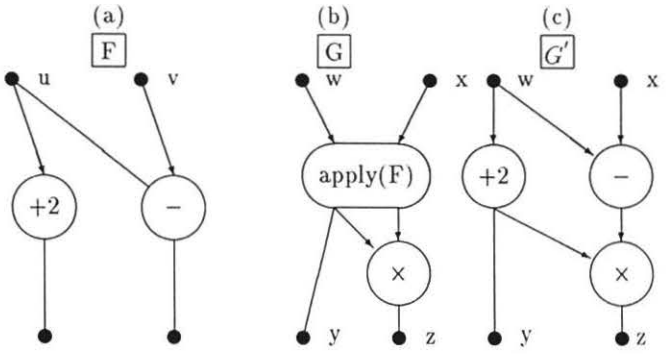

Figura 3: Ativação de função. (a) função F; (b) chamada da função F; (c) grafo após a substituição.

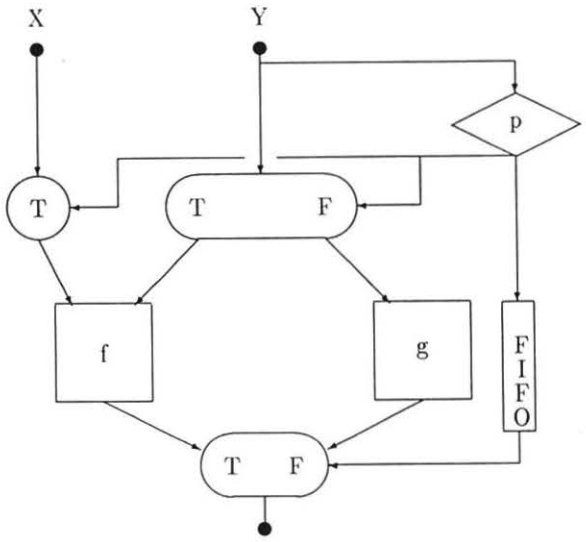

Figura 4: nó intercalador

num nó especial que implementa uma estrutura de fila. O nó intercalador é responsável pela sincronização entre fichas-resultado e as fichas oriundas dessa fila (figura 4).

O modelo resultante é mais robusto que o anterior, permitindo encadeamento e iteração, mas não a recursão, pois ainda não se consegue distinguir instâncias diferentes de um mesmo grafo.

A introdução de recursividade de cauda ("tail recursion"), uma forma restrita de recursão, agora é possível. Como conseqüência, não existe mais a possibilidade da representação direta de iterações que devem agora ser escritas na forma recursiva equivalente. A implementação deste tipo de recursão requer um novo campo na ficha, denominado ativação, para a identificação unívoca do seu contexto de execução.

O novo formato da ficha é:

< valor, ativação, aresta destino >

Um simulador para o estudo de modelos estáticos é proposto por [11].

3.3 O Modelo Dinâmico

O modelo **dinâmico** [12], [13] representa coerentemente iteração, encadeamento e recursão, implicando na ocorrência de fichas com arestas destino idênticas dentro de um mesmo estado. A separação entre essas fichas é conseguida através de um novo componente, denominado rótulo.

Portanto, o formato da ficha no modelo dinâmico é:

< valor, rótulo, aresta destino >.

O rótulo por sua vez é composto por
< ativação, laço, índice >, onde ativação separa os contextos de execução, laço separa as iterações de um comando repetitivo e índice separa os elementos de estruturas de dados.

Logo, há novas regras de disparo para os nós:

- um nó está habilitado se, e somente se, existirem fichas com o mesmo rótulo em todas suas arestas de entrada.
- qualquer conjunto de nós habilitados pode ser disparado simultaneamente.
- disparar um nó implica em remover um conjunto completo de fichas de mesmo rótulo das arestas de entrada, e produzir um conjunto coerente de fichas-resultado cujos campos de valor e rótulo são determinados pelo tipo de operação executada.

O modelo dispensa a utilização do operador intercalador e permite o disparo de um nó mesmo quando existem fichas nas suas arestas de saída.

Para a implementação de comandos repetitivos, são introduzidos operadores especiais para o manuseio dos campos da ficha relacionados ao rótulo, tais como:

- EL : um operador identidade que estabelece um novo contexto para a iteração e atribui ao componente laço de sua ficha-resultado o valor 1.
- SL : um operador identidade que restaura os componentes do rótulo da ficha, de acordo com o contexto externo ao laço.

Para gerenciar as demais possíveis mudanças de contexto num programa são utilizados operadores semelhantes.

A principal vantagem deste modelo vem de sua generalidade, que permite a representação direta dos principais conceitos de programação existentes (funções, iterações, recursão), sem comprometer as características de assincronismo e controle distribuído da computação.

No entanto, a manipulação eficiente de estruturas de dados, tais como vetores, matrizes e árvores neste modelo não é fácil. Normalmente, essa manipulação implica numa nova cópia da estrutura para cada operação realizada, com um impacto direto sobre o desempenho e necessidades do programa. Gostelow [14] aborda este problema e propõe alternativas para solucioná-lo.

4 Leituras Complementares

Alguns dos aspectos teóricos de computação paralela mais estudados estão relacionados ao não-determinismo, à prova de finitude e à correção de algoritmos concorrentes [15], [16], [17], [18], [19], [20], [21], [22].

Linguagens de programação, englobando o conceito de atribuição única ("single-assignment") às variáveis, têm sido propostas para facilitar a programação de máquinas paralelas [23], [24], [25], [26], [27]. Relatos sobre a implementação de algumas destas linguagens aparecem nos textos de [28], [29], [30] e [31].

A maior parte do trabalho inicial na área de fluxo de dados originou-se no grupo de Dennis [7], [9], [32], no MIT, onde foi proposto, por exemplo, o modelo estático. O modelo dinâmico foi amadurecido em trabalhos desenvolvidos simultânea e independentemente na Universidade de Manchester e na Universidade da Califórnia [33]. O trabalho do grupo de Manchester [34], [35], [36], [37], [38], deu origem a um protótipo que se tornou operacional em 1981 [39], [40]. Atualmente as pesquisas desse grupo têm-se voltado para o problema de gasto excessivo de memória [41], [42], [43], [44], [45].

Teoricamente, a capacidade computacional de máquinas de fluxo de dados é diretamente proporcional ao número de processadores empregados na sua construção. No entanto, os resultados obtidos demonstram que tal não ocorre [39], [46], embora pesquisas recentes demonstrem que a melhoria do desempenho observado é possível [47].

5 Conclusão

Neste texto apresentamos um resumo a respeito dos principais modelos de fluxo de dados existentes. As pesquisas nesta área são importantes principalmente para a identificação do modelo computacional paralelo que melhor se aplica à construção de máquinas de propósito geral.

A resposta a esta questão não implica necessariamente na designação do modelo de von Neumann como ultrapassado, podendo levar ao desenvolvimento de modelos computacionais razoáveis e eficientes. No futuro, modelos híbridos [6], [48] talvez sejam viáveis, e pesquisas nesta área estão em andamento.

O número de implementações de modelos de computação paralela, inclusive fluxo de dados, tem crescido [49], mas ainda não há um protótipo que seja completo o suficiente para sua utilização efetiva. Todos enfrentam problemas de manipulação de estruturas de dados, de recuperação de erros, de gerência de término de processos (conseqüência do assincronismo excessivo) e de identificação do paralelismo exibido pelos algoritmos.

Referências

[1] Backus, J., "Can Programming be liberated from the von Neumann Style? A Functional Style and its Algebra of Programs", Communications of the ACM, 21 (8), pag. 613-641, Agosto de 1978.

[2] Miller, R. E., "A Comparison of Some Theoretical Models of Parallel Computation", IEEE Trans. on Computers, C-22 (8), pag. 710-717, Agosto de 1973.

[3] Treleaven, P. C., Brownbridge, D. R., Hopkins e Richard P., "Data-driven and Demand-driven Computer Architecture", Computing Surveys, 14 (1), pag. 93-143, Março de 1982.

[4] Magó, G. A., "A Network of Microprocessors to Execute Reduction Languages, Part I", Int. J. of Computer and Info. Sciences, 8 (5), pag. 349-385, 1979.

[5] Magó, G. A., "A Network of Microprocessors to Execute Reduction Languages, Part II", Int. J. of Computer and Info. Sciences, 8 (6), pag. 435-471, 1979.

[6] Watson, I., Watson, P. e Woods, V., "Parallel Data Driven Graph Reduction", Fifth Generation Comp. Arch., Proc. of the 10 th World Computer Congress, IFIP 1986, pag. 203-219.

[7] Dennis, J. B., "Models of Data Flow Computation", in Control Flow and Data Flow: Concepts of Distributed Programming, NATO ASI Series, Vol. F14, Springer-Verlag, 1985, pag. 346-354.

[8] Kavi, K. M., Buckles, B. P. e Bhat, U. N., "A Formal Definition of Data Flow Graph Models", IEEE Trans. on Computers, C-35 (11), pag. 940-948, Novembro de 1986.

[9] Dennis, J. B., "Static Data Flow Computation", in Control Flow and Data Flow: Concepts of Distributed Programming, NATO ASI Series, Vol. F14, Springer-Verlag, 1985, pag. 355-363.

[10] Tanenbaum, A. S., "Structured Computer Organization", Prentice-Hall, 1984.

[11] Dennis, J. B., "VIM: An Experimental Computer System to Support General Functional Programming", in Control Flow and Data Flow: Concepts of Distributed Programming, NATO ASI Series, Vol. F14, Springer-Verlag, 1985, pag. 370-381.

[12] Arvind, Kathail, V. e Pingali, K., "A Dataflow Architecture with Tagged Tokens", MIT Laboratory for Computer Science, Int. Report MIT/LCS/TM-174, Setembro de 1980.

[13] Dettner, R., "Dataflow at MIT", Eletronics & Powers, 32 (8), pag. 570-571, Agosto de 1986.

[14] Gostelow, K. P. e Thomas, R. E., "A View of Dataflow", National Computer conference - AFIPS NCC , 48, pag. 1-8, Junho de 1979.

[15] Anderson, J. P., "Program Structures for Parallel Processing", Communications of the ACM, 8 (12), pag. 786-788, Dezembro de 1965.

[16] Karp, R. e Miller, R., "Properties of a Model for Parallel Computations: determinacy, termination, queueing", Siam J. Applied Math., 14 (6), pag. 1390-1411, Novembro de 1966.

[17] Hoare, C. A. R., "An Axiomatic Basis for Computer Programming", Communications of the ACM, 12 (10), pag. 576-580, Outubro de 1969.

[18] Dijkstra, E. W., "Guarded Commands, Nondeterminacy and Formal Derivation of Programs", Communications of the ACM, 18 (8), pag. 453-457, Agosto de 1975.

[19] Dijkstra, E. W., "A Discipline of Programming", Prentice-Hall, 1976.

[20] Hansen, P. B., "Distributed Processes: A Concurrent Programming Concept", Communications of the ACM, 21 (11), pag. 934-941, Novembro de 1978.

[21] Hoare, C. A. R., "Communicating Sequential Processes", Communications of the ACM, 21 (8), pag. 666-677, Agosto de 1978.

[22] Kierburtz, R. B. e Silberschatz, A., "Comments on Communicating Sequential Processes", ACM Transactions on Programming Languages and Systems, 1 (2), pag. 218-225, Outubro de 1979.

[23] Tesler, L. G. e Enea, H. J., "A Language Design for Concurrent Processes", AFIPS, Spring Joint Conference, 32, pag. 403-408, 1968.

[24] Chamberlin, D. D., "The "single-assignment" approach to parallel processing", Fall Joint Computer Conference, 39, pag. 263-269, 1971.

[25] Comte, Durrieu, Gelly, Plas e Syre, "Parallelism, Control and Synchronization Expressions in a Single Assignment Language", ACM Sigplan Notices, 13 (1), pag. 25-33, Janeiro de 1978.

[26] Dennis, J. B., "Functional Programming for Data Flow Computation", in Control Flow and Data Flow: Concepts of Distributed Programming, NATO ASI Series, Vol. F14, Springer-Verlag, 1985, pag. 364-369.

[27] Gajski, D. D., Padua, D. A., Kuck, D. J. e Kuhn, R. H., "A Second Opinion on Data Flow Machines and Languages", IEEE Computer, 15 (2), pag. 58-70, Fevereiro de 1982.

[28] Ashcroft, E. A. e Wadge, W. W., "Lucid, a Nonprocedural Language with Iteration", Communications of the ACM, 20 (7), pag. 519-526, Julho de 1977.

[29] Bush, V. J., "A Data Flow Implementation of Lucid", M.Sc. Thesis, Dep. of Computer Science, Univ. of Manchester, Outubro de 1979.

[30] McGraw, J. et. al., "SISAL - Streams and Iteration in a Single Assignment Language", Language Reference Manual, ver. 1.0, Lawrence Livermore National Lab., 1983.

[31] Wadge, W. W. e Ashcroft, E. A., "Lucid, the Dataflow Programming Language", Academic Press, 1985.

[32] Dennis, J. B., "Data Flow Supercomputers", IEEE Computer, 13 (11), pag. 48-56, Novembro de 1980.

[33] Arvind, Gostelow, K. P. e Plouffe, W., "An Asynchronous Programming Language and Computing Machine", Tech. Report, Dep. of Information and Computer Science, Univ. of California, Irvine, Dezembro de 1978.

[34] Gurd, J., Watson, I. e Glauert, J., "A Multilayered Data Flow Computer Architecture", Internal Report, Dep. of Computer Science, Univ. of Manchester, Julho de 1978.

[35] Gurd, J. e Watson, I., "Data Driven System for High Speed Parallel Computing - part 1: Structuring software for parallel execution", Computer Design, 19 (6), pag. 91-100, Junho de 1980.

[36] Gurd, J. e Watson, I., "Data Driven System for High Speed Parallel Computing - part 2: Hardware design", Computer Design, 19 (7), pag. 97-106, Julho de 1980.

[37] Catto, A. J., "Nondeterministic Programming in a Dataflow Environment", Ph.D. Thesis, Dep. of Computer Science, Univ. of Manchester, Junho de 1981.

[38] Watson, I. e Gurd, J., "A Practical Data Flow Computer", IEEE Computer, 15 (2), pag. 51-57, Fevereiro de 1982.

[39] Watson, I. e Gurd, J., "Preliminary Evaluation of a Prototype Dataflow Computer", Proc. of the 9 th World Computer Congress, IFIP 1983, pag. 545-551.

[40] Gurd, J. R., Kirkham, C. C. e Watson, I., "The Manchester Prototype Dataflow Computer", Communications of the ACM, 28 (1), pag. 34-52, Janeiro de 1985.

[41] Sargeant, J. e Kirkham, C. C., "Stored Data Structures on the Manchester Data Flow Machine", Computer Architecture News, 14 (2), pag. 235-242, Junho de 1986.

[42] Kawakami, K. e Gurd, J. R., "Scalable Data Flow Structure Store", Computer Architecture News, 14 (2), pag. 243-250, Junho de 1986.

[43] Gaudiot, J. L., "Methods for Handling Structures in a Data Flow System", Computer Architecture News, 14 (2), pag.352-358, Junho de 1986.

[44] Gaudiot, J. L., "Structure Handling in Data Flow Systems", IEEE Trans. on Computers, C-35 (6), pag. 489-502, Junho de 1986.

[45] Ruggiero, C. A., "Throttle Mechanisms for the Manchester Dataflow Machine", Tech. Rep. Series, UMCS-87-8-1, Ph.D Thesis, Dep. of Computer Science, Univ. of Manchester, Julho de 1987.

[46] Granski, M., Koren, I. e Silberman, G. M., "The Effect of Operation Scheduling on the Performance of a Dataflow Computer", IEEE Trans. on Computers, C-36 (9), pag. 1019-1029, Setembro de 1987.

[47] Ghosal, D. e Bhuyan, L.W., "Analytical and Architectural Modifications of a Data Flow Computer", Computer Architecture News, 15 (2), pag. 81-89, 1987.

[48] Buehrer, R. e Ekanadham, K., "Incorporating Data Flow Ideas into von Neumann Processors for Parallel Execution", IEEE Trans. on Computers, C-36 (12), pag. 1515-1522, Dezembro de 1987.

[49] Srini, V. P., "An Architectural Comparison of Data Flow Systems", IEEE Computer, 19 (3), pag. 68-88, Março de 1986.