ARQUITETURAS DIRIGIDAS PELO FLUXO DE DADOS

H. Piñón Arias; A. F. Castelo Branco; S. L. Olivier; A. J. Catto
Departamento de Ciência da Computação
Universidade Estadual de Campinas

SUMÁRIO

Apresenta-se uma revisão atualizada, embora não exaustiva, das arquiteturas dirigidas pelo fluxo de dados mais representativas. São descritas a máquina estática do MIT (Dennis) e a de frente de onda do Technion entre as estáticas, e a de Manchester, a máquina dinâmica do MIT (Arvind) e a SIGMA1 entre as dinâmicas. Analisam-se também as características básicas destas arquiteturas e indicam-se temas de pesquisa atuais nesta área.

ABSTRACT

An up-to-date but non exhaustive review of distinctive dataflow architectures is presented. Dennis' machine and Technion's wavefront array are described amongst the static dataflow computers, and the Manchester, Arvind's and SIGMA1 machines amongst the dynamic ones. The basic characteristics of these architectures are analysed and current research themes in this field are pointed out.

1. INTRODUÇÃO

Entre as alternativas consideradas para a construção de sistemas computacionais distribuídos tem sido dada particular atenção aos computadores não convencionais e, dentre estes, àqueles dirigidos pelo fluxo de dados [1], [2], [3], [4], [5], [6].

Nas máquinas de fluxo de dados, uma instrução está *habilitada*, e pode ser executada, assim que seus operandos estiverem disponíveis. Dessa forma, muitas instruções podem estar habilitadas simultaneamente, dispensando o recurso ao contador de instruções. O fluxo de controle é unicamente definido pelas dependências entre instruções, o que o torna apenas parcialmente determinado e permite que a execução de um programa seja altamente paralela e assíncrona.

A computação dirigida pelo fluxo de dados é modelada recorrendo-se à representação de programas sob a forma de grafos orientados, onde os nós representam instruções e as arestas indicam as relações produtor-consumidor existentes entre elas. Os dados são modelados como fichas que percorrem as arestas do grafo e são produzidas e consumidas pelos nós [7], [8], [9]. São dois os principais modelos baseados nesses conceitos: *estático* e *dinâmico* [5], [10], [11], [12], [13].

No modelo estático um nó está habilitado se existir uma ficha em cada uma de suas entradas e não existir ficha alguma em qualquer de suas saídas. Esse modelo é capaz de suportar a avaliação de expressões aritméticas, condicionais e repetitivas, além de uma forma restrita de funções recursivas.

No modelo dinâmico as fichas são rotuladas de acordo com o contexto a que pertencem. Isto permite que um mesmo nó torne-se habilitado simultaneamente por vários conjuntos de fichas de contextos distintos, eliminando-se a restrição imposta pelo modelo estático ao estado das arestas de saída dos nós habilitados. Assim, torna-se possível um amplo suporte à implementação de algoritmos recursivos, ainda que à custa do aumento do tamanho das fichas.

O que torna atraente um modelo computacional como o de fluxo de dados é a possibilidade de se extrair o máximo de paralelismo de um algoritmo, devida à distribuição natural do controle, à inexistência de memória compartilhada e à facilidade de programação, por exemplo, através de linguagens funcionais [10], [14], [15], [16], [17], [18], [19].

As seções seguintes examinam algumas arquiteturas representativas de cada modelo, analisam problemas encontrados durante o desenvolvimento dos respectivos protótipos e apontam linhas de pesquisa atuais e promissoras nesta área.

2. ARQUITETURAS ESTÁTICAS

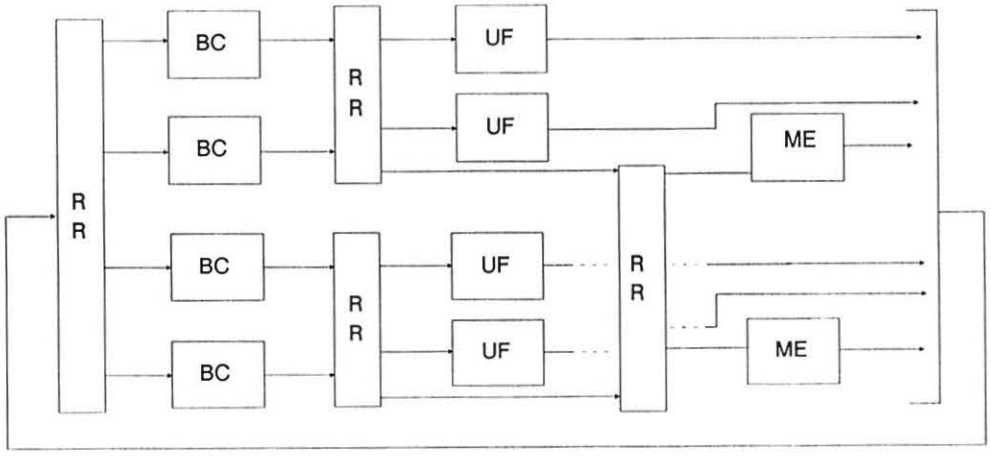
Estas arquiteturas implementam o modelo de fluxo de dados estático, onde, em cada instante, um nó pode estar habilitado dentro de apenas um único contexto. São exemplos desta classe a arquitetura estática do MIT [20] e a arquitetura de frente de onda do Technion [21].

2.1 A ARQUITETURA ESTÁTICA DO MIT

Dennis [20] propõe um sistema multiprocessador constituído por *blocos de células, unidades funcionais* e *memórias de estruturas*, interligados por *redes roteadoras* [Fig.1].

O código correspondente a um programa é distribuído inicialmente entre os blocos de células, que são responsáveis pelo reconhecimento das instruções habilitadas, pela execução das mais simples e pela transferência das mais complexas para alguma unidade funcional disponível. Todas as comunicações ocorrem através de redes roteadoras. Os pacotes de resultados produzidos pelos blocos de células e pelas unidades funcionais são distribuídos pelas redes de roteamento para os blocos de células, realimentando o processo.

Buscando melhorar o desempenho esperado dessa arquitetura, foi proposta a divisão da capacidade de armazenamento entre as memórias de programa, distribuídas entre os blocos de células, e memórias de estruturas, unidades independentes para a manutenção de dados estruturados em aplicações de grande porte.



RR ; Rede de Roteadores UF : Unidade Funcional

BC : Bloco de Celulas ME : Memoria de Estruturas

FIGURA 1: Arquitetura de Dennis Estatica

O objetivo é atingir uma taxa de reconhecimento e execução de instruções de 1 Mips nos blocos de células, levando a 1 GFlops no sistema como um todo.

2.2 ARQUITETURA DE FRENTE DE ONDA DO TECHNION

As arquiteturas sistólicas permitem a implementação em hardware de algoritmos de fluxo de dados com fins específicos. Essas arquiteturas recorrem a processadores dedicados para cada operação, conectados segundo um particular grafo de fluxo de dados e controlados sincronizadamente por um relógio central [22].

As arquiteturas de frente de onda representam uma tendência mais recente, que recorre a processadores microprogramáveis, com relógios locais, para permitir a reconfiguração dinâmica do grafo implementado e a gerência de sua execução segundo as regras de fluxo de dados [23].

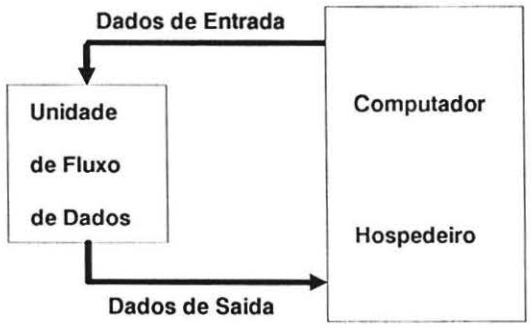


FIGURA 2

Na arquitetura proposta pelo grupo do Technion [21], o grafo de fluxo de dados resultante da compilação de um programa na linguagem VAL [14] é mapeado sobre uma grade de processadores dirigidos pelos dados. Esta grade é constituída por processadores hexagonais, ligados aos seus vizinhos mais próximos e agrupados em pastilhas VLSI padronizadas [24]. Pela interconexão de pastilhas segundo a mesma regra, pode-se expandir arbitrariamente a grade e associá-la como coprocessador a um sistema hospedeiro [Fig.2].

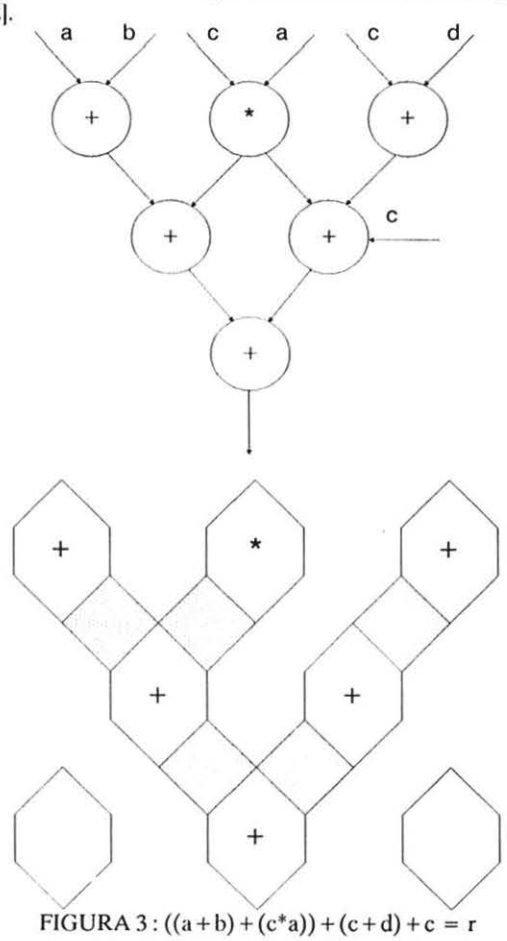


FIGURA 3 : ((a + b) + (c*a)) + (c + d) + c = r

A programação é realizada em quatro passos. Primeiro obtém-se o grafo de fluxo de dados associado ao programa fonte. Em seguida esse grafo é mapeado sobre uma grade virtual de processadores hexagonais, finita mas de tamamnho suficiente. Nesta fase cada nó do grafo é associado a um dos processadores dessa grade e as conexões entre estes são definidas de acordo com as arestas do grafo implementado [Fig.3]. O próximo passo consiste em partir a grade em um determinado número de segmentos de acordo com a quantidade de processadores que podem ser empacotados numa mesma pastilha. Finalmente, cada processador da grade real é microprogramado com as tarefas atribuídas aos processadores virtuais que ele implementa.

A grande vantagem desta abordagem é permitir a criação de hardware "sob medida", num processo semelhante à programação de EPROMs, mantendo baixos os custos finais.

3. ARQUITETURAS DINÂMICAS

Estas arquiteturas implementam o modelo de fluxo de dados dinâmico, que permite a habilitação simultânea de um nó em vários contextos distintos. São exemplos desta classe a máquina de Manchester [4], [13], [25], [26], [27], [28], [29], a máquina dinâmica do MIT [10], [30] e as máquinas desenvolvidas no ICOT [31], [32], [33].

3.1 MÁQUINA DE MANCHESTER

Esta arquitetura é estruturada como um anel circular composto por quatro módulos principais, conectado a um computador hospedeiro através de uma unidade para chaveamento de entrada

FIGURA 4 : Maquina de Manchester

e saída [Fig.4]. O anel constitui uma "pipeline" com múltiplos estágios, pela qual circulam fichas rotuladas representando os dados e seus respectivos contextos. Os módulos principais atuam de modo assíncrono e implementam *unidades de regulagem, emparelhamento, programa* e *processamento*.

A unidade de regulagem constitui uma fila destinada ao armazenamento temporário de fichas, de modo a equilibrar o ritmo com que estas são produzidas e consumidas pelo sistema.

A unidade de emparelhamento agrupa as fichas destinadas a uma mesma instrução binária. Implementada de forma pseudo-associativa, utiliza os campos de rótulo e destino da ficha para, através de hashing, localizar sua parceira em bancos de memória de acesso aleatório. Fichas destinadas a operações unárias passam livremente por este módulo.

Os pares de fichas criados pela unidade de emparelhamento buscam a respectiva instrução na unidade de programa através de seu endereço de destino.

A unidade de programa forma pacotes executáveis pela associação de argumentos e instruções e os remete à unidade de processamento, um grupo de microprocessadores microprogramáveis dispostos em paralelo. Um sistema de distribuição é responsável pela seleção de um processador livre para processar o pacote recebido. Um sistema de arbitração encarrega-se de encaminhar de volta ao anel as fichas produzidas como resultado pelos diversos processadores.

Não é possível melhorar indefinidamente o desempenho dessa máquina pela simples adição de novos elementos à unidade de processamento, uma vez que as demais unidades acabarão por limitar a velocidade no anel. Para aliviar esse problema o projeto original recebeu diversas extensões, tais como *funções de emparelhamento* [17], *memória para estruturas de dados* [34], [35] e um *controlador de paralelismo* [36]. Além disso, está em estudo um sistema dotado de múltiplos anéis interconectados, no qual todas as unidades são distribuídas [4], [13], [37].

3.2 A ARQUITETURA DINÂMICA DO MIT

Esta arquitetura consiste de *N unidades de processamento* e uma *rede de comunicação de pacotes NN* [Fig.5].

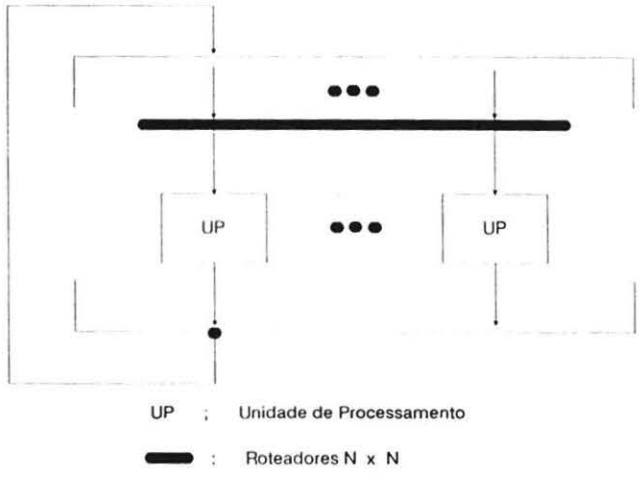


FIGURA 5 : Arquitetura dinamica do MIT

A rede de comunicação distribui o programa e os pacotes de dados entre as unidades de processamento, utilizando transmissão serial. Um pacote de dados é composto por um valor e um rótulo que identifica a atividade a que ele se destina. A distribuição do programa é feita através de uma *função de atribuição*, que mapeia rótulos em unidades de processamento. Para maior eficiência, instruções pertencentes a blocos coesos, tais como procedimentos ou laços, são alocados a processadores tão próximos quanto possível.

Cada unidade de processamento é organizada em seis seções: *de entrada, de emparelhamento, de programa, de serviço, de estruturas* e *de saída* [Fig.6]. Todo pacote destinado a uma unidade de

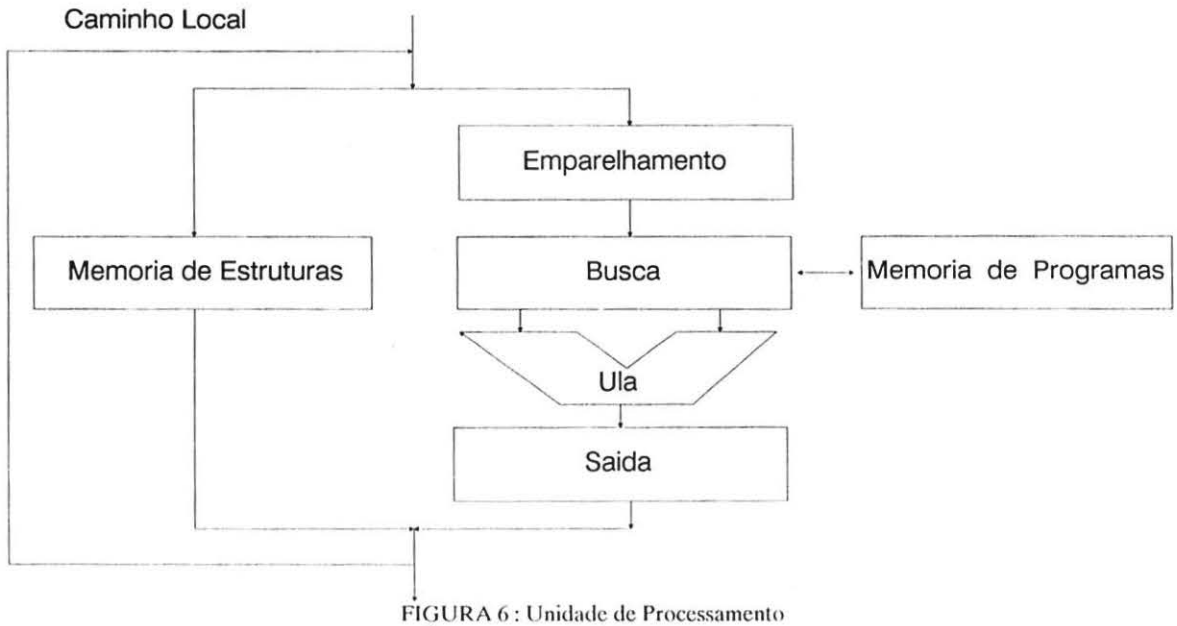


FIGURA 6 : Unidade de Processamento

processamento é recebido pela seção de entrada e, caso enderece uma instrução binária, é encaminhado à seção de emparelhamento. Esta retém o pacote recebido, a menos que já esteja armazenado um outro pacote de mesmo rótulo; nesse caso, ambos são encaminhados à seção de programa.

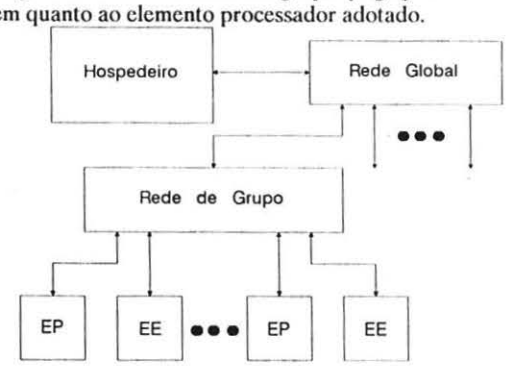
Na seção de programa os pacotes são associados à instrução correspondente, antes de ser executado na seção de serviço.

O encaminhamento dos pacotes de resultados é feito através da seção de saída, utilizando a função de atribuição para determinar a unidade de processamento de destino. Se o destino for a própria unidade, o pacote é transferido diretamente para a seção de entrada; caso contrário, é encaminhado à rede de comunicação.

Estruturas de dados mais complexas são armazenadas pela seção de estruturas, que é implementada através de memória de acesso aleatório.

3.3 SIGMA1

O ICOT japonês está desenvolvendo dois projetos de máquinas de fluxo de dados: SIGMA1 [32], [33] e uma máquina de inferência [31]. Apesar de apresentarem semelhanças de projeto, uma vez que ambos baseiam-se em uma rede de comunicação hierárquica à qual estão ligadas redes locais ou de grupo [Fig.7], esses sistemas diferem quanto ao elemento processador adotado.



EP : Elemento Processador.
EE : Elemento de Estruturas.

FIGURA 7: Arquitetura SIGMA-1

No SIGMA1 um grupo de processamento é constituído por quatro elementos processadores e quatro elementos de estrutura, associados por uma rede de grupo. As tarefas geradas por um bloco de código (procedimento, laço ou comando composto) é atribuída a um grupo. Cada um dos elementos processadores do grupo recebe uma cópia completa do bloco de código, de modo a ter acesso rápido a qualquer uma das instruções correspondentes.

A rede local é do tipo "crossbar" 10x10, implementada com processadores "bit slice" e empacotada numa pastilha VLSI, para obter resposta imediata e tempo de latência pequeno. A rede global é uma rede hierárquica de estágios múltiplos, implementado usando o mesmo chip.

As unidades do elemento processador da SIGMA1 estão acomodadas numa "pipeline" de dois estágios [Fig.8]. O registrador de entrada funciona como interface entre a rede local e o elemento processador. A unidade de emparelhamento recebe fichas rotuladas e, através de um esquema de "hashing" encadeado, agrupa os operandos destinados a instruções binárias, enquanto deixa passar aqueles destinados a instruções unárias.

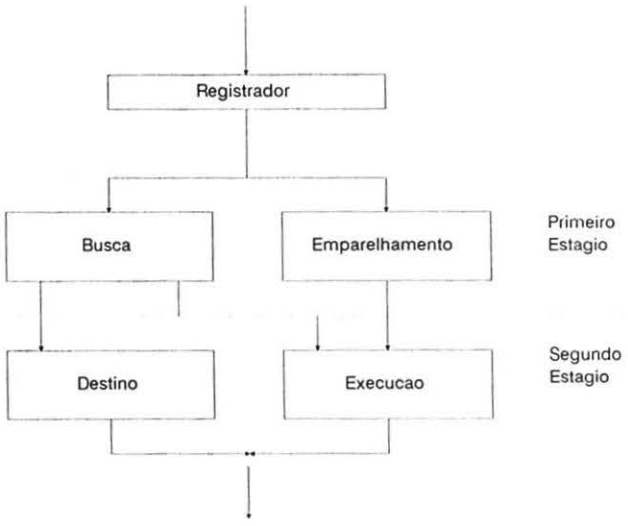


FIGURA 8 : Pipeline do SIGMA-1

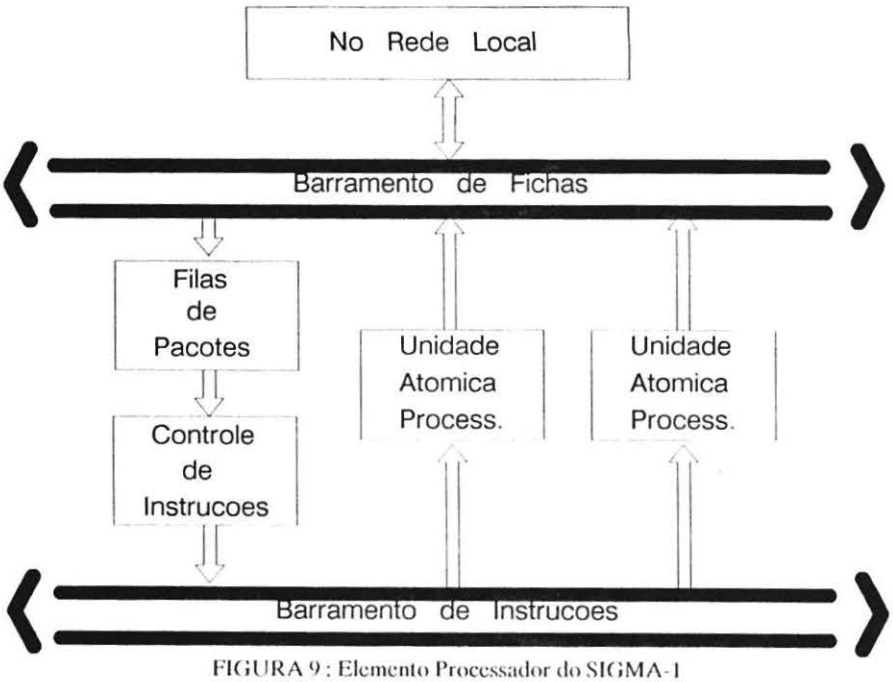


FIGURA 9 : Elemento Processador do SIGMA-1

unidade de busca recebe da rede o identificador de um nó, calcula o respectivo endereço e recupera a instrução correspondente. O código da instrução é enviado à unidade de execução, enquanto os até três endereços de destino dos resultados são passados à unidade de destino.

A unidade de execução tem um tempo de ciclo de 80 ns, trabalha em forma síncrona e pode atingir uma taxa de operação da ordem de 4 Mflops. O objetivo a ser alcançado pelo sistema é 100 Mflops.

A unidade de estruturas é uma memória dedicada ao armazenamento de EstruturasI [38], associada a uma unidade de controle para gerenciar a memória livre e as listas de espera.

A máquina de inferência para AndParallel Prolog e OrParallel Prolog apresenta elementos processadores dotados de maior paralelismo, através da incorporação de uma fila de pacotes e de várias unidades de processamento atômicas [Fig.9]. Os grupos são formados por oito elementos processadores e nove memórias de estruturas. O tempo de ciclo dessa máquina é de 250 ns, desejando-se chegar a 1 Mhups (head unifications per second), com 64 elementos processadores.

4. AVALIAÇÃO

Para Dennis [39] uma das principais qualidades de um sistema distribuído de propósito geral é a transparência do processo de descentralização do controle. Gostelow e Thomas [40] também enumeram alguns objetivos principais, comuns a muitos dos projetos desenvolvidos nesta área:

(a) promover a distribuição eficiente do controle entre um grande número de processadores;

(b) exibir modularidade, a fim de permitir expansão linear do sistema mediante o simples acréscimo de unidades;

(c) ser tolerante a falhas;

(d) possuir configuração transparente ao usuário;

(e) independer de hardware sofisticado para obter alto desempenho;

(f) ter capacidade multiusuário.

Algumas dessas metas, como a transparência na distribuição do controle, por exemplo, estão sendo amplamente alcançadas pelos projetos examinados. Outras, como a expansibilidade, eficiência e tolerância a falhas, têm sido atingidas em apenas alguns casos. Esta seção discute vantagens e desvantagens e avalia os desdobramentos dessas propostas.

Dos aspectos analisados, a questão da eficiência relativamente modesta das máquinas de fluxo de dados é o mais controverso. Quando se trata de avaliar essas máquinas, é preciso levar em conta a diferença nos conceitos de Mips e Mflops e que protótipos acadêmicos, construídos principalmente para demonstrar a viabilidade de um conceito, estão sendo comparados a uma tecnologia industrialmente madura e refinada [33], [41], [42], [43].

As linguagens funcionais [44], que permitem expressar os algoritmos de forma elegante e sintética e tornam transparente para o programador a implementação do paralelismo, têm dominado a pesquisa na área de programação de máquinas de fluxo de dados [7], [10], [14], [16], [18], [25], [45]. Sua tradução para grafos de fluxo de dados, a linguagem de baixo nível dessas máquinas, é conseguida de forma natural, devida principalmente à precisão de sua definição semântica e à ausência de efeitos colaterais [46].

Devido à grande semelhança entre os respectivos modelos, máquinas de fluxo de dados também têm sido consideradas para a implementação de linguagens de programação lógica, cujos programas baseiam-se na definição e busca de objetivos [31].

A implementação dessas linguagens traz consigo o problema da *granularidade*, o tamanho mínimo dos módulos de código distribuídos entre os processadores do sistema. A granularidade do paralelismo é usualmente classificada como *grossa* (processos independentes), *média* (rotinas, laços ou expressões) ou *fina* (instruções) [47]. Todas as arquiteturas examinadas implementam granularidade fina ou média [5], [6]. Uma das desvantagens de se usar granularidade fina é a sobrecarga de comunicação que ela acarreta. A granularidade média, por sua vez, além de dificultar a extração do paralelismo, elimina o assincronismo na execução do bloco de código correspondente [26], [29], [48], [49], [50], [51].

Diretamente associado à granularidade está o problema do mapeamento do grafo de fluxo de dados sobre a arquitetura física. Por razões de eficiência, é necessário levar em conta o grau de localidade do algoritmo, para reduzir as comunicações entre as unidades do sistema, e, ao mesmo tempo, balancear a carga entre os elementos de processamento. Estes dois requisitos são antagônicos e inerentes a qualquer sistema distribuído [5], [49], [50].

Os ensaios com protótipos e as simulações realizadas pelos diferentes grupos de pesquisa demonstram que a sobrecarga gerencial para a execução de um programa num sistema de fluxo de dados é realmente maior do que numa arquitetura convencional [5], [29], [52], [53], [54]. Isto se deve basicamente a:

(a) maior número de referências à memória, devido à inexistência de registradores;

(b) maior número de instruções associadas à execução de laços e chamadas de procedimentos;

(c) maior quantidade de memória para armazenar a mesma informação devida ao rótulo das fichas.

Entre as alternativas consideradas para solução desses problemas estão o uso de instruções mais complexas e fichas estacionárias *(sticky tokens)*. No primeiro caso, o objetivo é agrupar instruções que, devido à topologia do grafo, devem ser executadas seqüencialmente [33], [55]. A segunda alternativa foi considerada inicialmente em Manchester através de *funções de emparelhamento* que permitem, por exemplo, incrementar ou decrementar uma variável de controle diretamente na unidade de emparelhamento [17], [26], [33].

Comprovou-se também a existência de um limite para a expansão útil dessas máquinas, decorrente de um *efeito saturação* [29], [49], [53]. Este pode ser devido ao tipo e largura de banda da rede de comunicação entre os elementos da arquitetura, à granularidade adotada ou ao aparecimento de bolhas na "pipeline" provocadas pelo processo de emparelhamento de fichas [5], [29], [50]. A solução desse problema dependerá provavelmente da adoção de algoritmos mais eficientes para mapeamento de código e alocação de recursos.

Verificou-se ainda a necessidade de um mecanismo eficiente para o tratamento de estruturas de dados regulares. A satisfação do requisito de atribuição única *(single-assignment)* das linguagens funcionais implicou, nos primeiros modelos propostos, na cópia completa de cada estrutura a cada referência, mesmo nos casos de modificações mínimas. A primeira solução para este problema foi proposta pelo grupo de Manchester, através do uso de funções de emparelhamento específicas, capazes de permitir a atualização seletiva de uma estrutura de dados na unidade de emparelhamento, sem o ônus representado pela sua duplicação [17], [26]. Essa proposta evoluiu posteriormente para a inclusão no anel de uma memória dedicada ao armazenamento de estruturas de dados [34], [35]. Solução semelhante foi adotada para a arquitetura dinâmica do MIT, através de estruturasI armazenadas em unidades de memória específicas [38]. Neste caso, o acesso às estruturas é feito através de ponteiros, existindo indicadores que permitem a busca eficiente e a recuperação do espaço liberado por estruturas fora-de-uso. Embora existam várias outras propostas [56], [57], uma solução adequada para o problema da representação e tratamento de estruturas de dados é ainda uma questão essencialmente aberta.

Mesmo com o desafogamento representado pela inclusão de memórias de estruturas nessas máquinas, continua existindo a necessidade de evitar a saturação das memórias temporárias e dos rótulos das fichas, responsáveis por uma inevitável degradação do desempenho em programas de maior porte. As soluções apontadas para este problema incluem a restrição do assincronismo suportado pela máquina, com a conseqüente limitação do número de processos habilitados para execução em cada instante [36], [54], [58], e o reaproveitamento dos rótulos de contextos já fora de atividade [17].

Finalmente, há a questão do tratamento de erros de execução, que se torna muito complicado em máquinas distribuídas assíncronas de propósito geral. Este assunto não foi abordado adequadamente por nenhum dos projetos examinados.

5. CONCLUSÃO

As primeiras idéias a respeito de fluxo de dados surgiram há cerca de 15 anos. Desde então, vários projetos foram desenvolvidos e alguns protótipos construídos e avaliados [5]. Os resultados obtidos afastam algumas das dúvidas levantadas sobre esses sistemas, ao mesmo tempo em que reforçam a necessidade de maior investimento na pesquisa de alguns aspectos essenciais [59] e não permitem conclusões definitivas sobre o modelo computacional mais adequado à implementação de sistemas paralelos de propósito geral [60], [61], [62].

Enquanto essas questões não recebem resposta adequada, várias idéias surgidas nos projetos de fluxo de dados vão sendo aproveitadas por áreas de perfil mais conservador [63]. Por outro lado, o estudo dos sistemas disponíveis, através de técnicas de modelamento, mostra que é possível aprimorar o desempenho originalmente exibido por essas arquiteturas e que a última palavra a seu respeito ainda não foi dita [37].

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Agerwala, T. & Arvind. Dataflow Systems. IEEE Computer, 15(2): 10-13, fev 82.

[2] Denning, P.J. Special Section on Computer Architecture, (Introduction). Communications of the ACM, 28(1): 67, jan 85.

[3] Dettner, R. Dataflow at MIT. Eletronics & Powers, 32(8): 570-571, ago 86.

[4] Gurd, J.R.; Kirkham, C.C. & Watson, I. The Manchester Prototype Dataflow Computer. Communications of the ACM, 28(1): 34-52, jan 85.

[5] Srini, V.P. An Architectural Comparison of Data Flow Systems. IEEE Computer, 19(3): 68-88, mar 86.

[6] Treleaven, P.C.; Brownbridge, D.R. & Hopkins, R.P. Data-Driven and Demand-Driven Computer Architecture. ACM Computing Surveys, 14(1): 93-143, mar 82.

[7] Ackerman, W.B. Data Flow Languages. IEEE Computer, 15(2): 15-25, fev 82.

[8] Davis, A. & Keller, R.M. Data Flow Program Graphs. IEEE Computer, 15(2): 26-41, fev 82.

[9] Kavi, K.M.; Buckles, B.P. & Bhat, U.N. A Formal Definition of Data Flow Graph Models. IEEE Transactions on Computers, C35(11): 940-948, nov 86.

[10] Arvind; Gostelow, K.P. & Plouffe, W. An Asynchronous Programming Language and Computing Machine. Technical Report. Department of Information and Computer Science, University of California, Irvine, dez 78.

[11] Buzato, L.E.; Calsavara, C.M.F.R. & Catto, A.J. Modelos Computacionais de Fluxo de Dados.

[12] Dennis, J.B. Data Flow Supercomputers. IEEE Computer, 13(11): 48-56, nov 80.

[13] Gurd, J.R.; Watson, I. & Glauert, J. A Multilayered Data Flow Computer Architecture. Technical Report. Department of Computer Science, University of Manchester, jul 78.

[14] Ackerman, W.B. & Dennis, J.B. VAL A Value Oriented Algorithmic Language. Preliminary Reference Manual. Technical Report TR218. Laboratory for Computer Science, MIT, jun 79.

[15] Ashcroft, E.A. & Wadge, W.W. Lucid, a Nonprocedural Language with Iteration. Communications of the ACM, 20(7): 519-526, jul 77.

[16] Bush, V.J. A Data Flow Implementation of Lucid. M.Sc. Dissertation. Department of Computer Science, University of Manchester, out 79.

[17] Catto, A.J. Nondeterministic Programming in a Dataflow Environment. Ph.D. Thesis. Department of Computer Science, University of Manchester, jun 81.

[18] 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, p.364-369.

[19] Wadge, W.W. & Ashcroft, E.A. Lucid, the Dataflow Programming Language. Academic Press, 1985.

[20] 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, p.355-363.

[21] Mendelson, B. & Silberman, G.M. Mapping Data Flow Programs on a VLSI Array of Processors. Computer Architecture News, 15(2): 72-80, 1987.

[22] Fortes, J.A.B. & Wah, B.W. Systolic Arrays From Concept to Implementation. IEEE Computer, 20(7): 12-17, jul 87.

[23] Kung, S.Y.; Lo, S.C.; Jean, S.N. & Hwang, J.N. Wavefront Array Processors Concept and Implementation. IEEE Computer, 20(7): 18-33, jul 87.

[24] Koren, I. & Peled, I. The Concept and Implementation of Data-Driven Processor Arrays. IEEE Computer, 20(7): 102-103, jul 87.

[25] Gurd, J.R. & Watson, I. Data Driven System for High Speed Parallel Computing Part 1: Structuring software for Parallel Execution. Computer Design, 19(6): 91-100, jun 80.

[26] Gurd, J.R. & Watson, I. Data Driven System for High Speed Parallel Computing Part 2: Hardware Design. Computer Design, 19(7): 97-106, jul 80.

[27] Gurd, J.R. The Manchester Dataflow Machine. Future Generation Computer Systems. North-Holland, 1985, p.201-212.

[28] Watson, I. & Gurd, J.R. A Practical Data Flow Computer. IEEE Computer, 15(2): 51-57, fev 82.

[29] Watson, I. & Gurd, J.R. Preliminary Evaluation of a Prototype Dataflow Computer. Proceedings of the 9th World Computer Congress, IFIP 83, p.545-551.

[30] Arvind; Kathail, V. & Pingali, K. A Dataflow Architecture with Tagged Tokens. Technical Report TM174. Laboratory for Computer Science, MIT, set 80.

[31] Ito, N.; Kishi, M.; Kuno, E. & Rokusawa, K. The Dataflow-based Parallel Inference Machine To Support Two Basic Languages in KL1. In: Fifth Generation Computer Architectures, ed. J.V. Woods. Elsevier, IFIP, 1986, p.123-145.

[32] Shimada, T.; Hiraki, K. & Nishida, K. An Architecture of a Data Flow Machine and its Evaluation. IEEE COMPCON-Spring, 1984, p.486-490.

[33] Shimada, T.; Hiraki, K.; Nishida, K. & Sekiguchi, S. Evaluation of a Prototype Data Flow Processor of the SIGMA1 for Scientific Computation. IEEE, 1986, p.226-234.

[34] Kawakami, K. & Gurd, J.R. A Scalable Data Flow Structure Store. ACM Computer Architecture News, 14(2): 243-250, jun 86.

[35] Sargeant, J. & Kirkham, C.C. Stored Data Structures on the Manchester Data Flow Machine. Computer Architecture News, 14(2): 235-242, jun 86.

[36] Ruggiero, C.A. Throttle Mechanisms for the Manchester Dataflow Machine. Phd Thesis. Technical Report Series, UMCS8781. Department of Computer Science, University of Manchester, jul 87.

[37] Ghosal, D. & Bhuyan, L.W. Analytical and Architectural Modifications of a Data Flow Computer. ACM Computer Architecture News, 15(2): 81-89, fev 87.

[38] Arvind & Thomas, R.E. I-Structures: An Efficient Data Type for Functonal Languages. Technical Report TM178. Laboratory for Computer Science, MIT, jun 80.

[39] 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, p.346-354.

[40] Gostelow, K.P. & Thomas, R.E. A View of Dataflow. National Computer Conference AFIPS NCC, 48: 18, jun 79.

[41] Dongarra, J.; Martin, J. & Lorton, J. Computer Benchmarking: Paths and Pitfalls. IEEE Spectrum, 24(7), jul 87.

[42] Hack, J.J. Peak vs Sustained Performance in Highly Concurrent Machines. IEEE Computer, 19(9): 11-19, set 86.

[43] Serlin, O. Mips, Drystones, and Other Tales. Datamation: 112-118, jun 86.

[44] 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): 613-641, ago 78.

[45] 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, p.370-381.

[46] Tesler, L.G. & Enea, H.J. A Language Design for Concurrent Processes. AFIPS, Spring Joint Conference, 32: 403-408, 1968.

[47] Mokhoff, N. Parallelism Breeds a New Class of Supercomputers. Computer Design, 26(?): 53-64, 15 mar 87.

[48] Arvind & Gostelow, K.P. The UInterpreter. IEEE Computer, 15(2): 42-49, fev 82.

[49] Chu, W.W.; Holloway, L.J.; Lan, M. & Efe, K. Task Allocation in Distributed Data Processing. IEEE Computer, 13(11): 57-69, nov 80.

[50] Cvetanivic, Z. The Effects of Problem Partitioning, Allocation and Granularity on the Performance of Multiple-Processor Systems. IEEE Transactions on Computers, C36(4): 421-432, abr 87.

[51] Gransky, M.; Koren, I. & Silberman, G. The Effect of Operation Scheduling on the Performance of a Data Flow Computer. IEEE Transactions on Computers, C36(9): 1019-1029, set 87.

[52] Gajski, D.D.; Padua, D.A.; Kuck, J. & Kuhn, R.H. A Second Opinion on Data Flow Machines and Languages. IEEE Computer, 15(2): 58-70, fev 82.

[53] Keller, S.R.M.; Lin, F.C.H. & Tanaka, J. Rediflow Multiprocessing. IEEE COMPCON-Spring, 1984, p.410-417.

[54] Sunahara, H. & Tokoro, M. On the Working Set Concept for Data-Flow Machines: Policies and their Evaluation. In: Fifth Generation Computer Architectures, ed. J.V. Woods. Elsevier, IFIP, 1986, p.147-161.

[55] Fisher, J.A. & O'Donnell, J.J. VLIW Machines: Multiprocesors We Can Actually Program. IEEE COMPCON-Spring, 1984, p.299-305.

[56] Gaudiot, J.L. Methods for Handling Structures in a Data Flow System. Computer Architecture News, 14(2): 352-358, jun 86.

[57] Gaudiot, J.L. Structure Handling in Data Flow Systems. IEEE Transactions on Computers, C35(6): 489-502, jun 86.

[58] Arvind & Culler, D.E. Managing Resources in a Parallel Machine. In: Fifth Generation Computer Architectures, ed. J.V. Woods. Elsevier, IFIP, 1986, p.103-121.

[59] Serlin, O. Parallel Processing: Fact or Fancy? Datamation: 93-105, dez 85.

[60] Miller, R.E. A Comparison of Some Theoretical Models of Parallel Computation. IEEE Transactions on Computers, C22(8): 710-717, ago 73.

[61] Treleaven, P.C. & Gouveia Lima, I. Future Computers: Logic, Data Flow, ..., Control Flow? IEEE Computer, 17(3): 47-58, mar 84.

[62] Watson, I.; Watson, P. & Woods, V. Parallel Data Driven Graph Reduction. Fifth Generation Computer Architecture, Proceedings of the 10th World Computer Congress, IFIP 86, p.203-219.

[63] Buehrer, R. & Ekanadham, K. Incorporating Data Flow Ideas into von Neumann Processors for Parallel Execution. IEEE Transactions on Computers, C36(12), dez 87.