Primeiros Resultados da Proto-Arquitetura a Fluxo de Dados Wolf

A. Garcia Neto* M. A. Cavenaghi†

Grupo de Instrumentação e Informática
Departamento de Física e Ciência dos Materiais
Universidade de São Paulo, Brasil

Resumo

Este artigo apresenta a arquitetura preliminar definida para estudar características desejáveis para o processador Wolf. O projeto Wolf propõe-se a implementar e estudar as características de um supercomputador de alta velocidade baseado no modelo de fluxo de dados de granularidade variável. Os primeiros resultados das simulações dessa arquitetura são também apresentados e comparados com dados conhecidos. Para contextualizar a proposta Wolf no contexto da pesquisa contemporânea em fluxo de dados, o artigo inclui uma resenha dos principais projetos europeus, americanos e japoneses.

Palavras-chave: Arquiteturas Paralelas, Implementação MIMD, Simulação de Sistemas, Fluxo de Dados, Granularidade Variável.

Abstract

This paper presents the architecture preliminarly defined to study the desirable characteristics for the Wolf processor. The Wolf project is a proposal for the implementation of a supercomputer based on the dataflow model with variable granularity. The first simulation results are presented and compared to known results. In order to place the proposal in context, a survey including many dataflow projects in Europe, North America and Japan is also included.

*alvaro@uspfsc.ifqsc.usp.ansp.br
†marcos@uspfsc.ifqsc.usp.ansp.br

1 Introdução

A arquitetura Wolf deriva da Máquina Multi-anel de Manchester (*Manchester Multi-ring Dataflow Machine - MMDM*) [10], incorporando muitas das lições aprendidas desde a definição da MMDM pelo grupo do Prof. Gurd na década de 80, aproveitando também os resultados de estudos indepentendes sobre a MMDM realizados em outras instituições.

A arquitetura Wolf evoluiu de um conjunto de premissas, algumas das quais comprovadas por pesquisas recentes, outras propostas pelos autores ou outros pesquisadores.

- Máquinas a fluxo de dados resolvem de forma elegante e eficiente diversos problemas típicos de arquiteturas paralelas. Alguns desses problemas são de difícil solução, como por exemplo, alocação de processos e distribuição de carga autonomos, linearidade da velocidade de processamento em relação ao número de processadores, contenção de memória e latencia de acesso.
- Máquinas dataflow são fáceis de programar, quando comparadas a outras arquiteturas MIMD, e podem expor e explorar o paralelismo intrínseco dos algoritmos sem a ajuda do programador.
- Máquinas a fluxo de dados possuem um mecanismo bem definido e eficiente para controlar e evitar a exposição de paralelismo excessivo e nocivo [16, 19].
- Processadores com arquitetura von Neumann são capazes de processar algoritmos de baixo paralelismo de forma mais eficiente do que máquinas a fluxo de dados [7, 12].
- Processadores com arquitetura SIMD são mais eficiente do que arquiteturas a fluxo de dados para explorar o paralelismo derivado de regularidades na estrutura dos dados.
- A filosofia RISC permite a implementação de processadores seqüenciais de alta eficiência [23].
- Arquiteturas a fluxo de dados são pródigas no uso de recursos de memória [7].
- A implementação do mecanismo de disparo de fluxo de dados dinâmico com circuitos de memória endereçáveis por conteúdo é complexa e onerosa em termos de área de pastilha de CI.

2 Arquiteturas a Fluxo de Dados

As arquiteturas a fluxo de dados tem evoluído a partir de uma intensa interação num conjunto relativamente pequeno de grupos de pesquisa. Esta seção coloca o desenvolvimento da arquitetura Wolf dentro do contexto do trabalho contemporâneo nesta área.

2.1 Pesquisa na Europa

2.1.1 Manchester

O grupo de Pesquisas de Manchester propos uma arquitetura a fluxo de dados dinâmica baseada em anel [10] mostrada na figura 1.

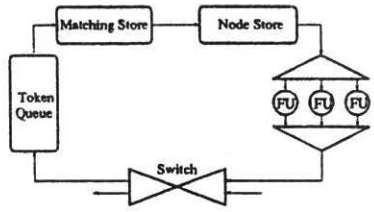

Figura 1: A arquitetura de Manchester

A MMDM é uma coleção de anéis conectados por uma rede de comutação. Dentro de cada anel há unidades para homogeneizar o tráfego (*Token Queue*), para implementar a regra de disparo e armazenar resultados intermediários (*Matching Unit*), para armazenar o grafo de dependencia de dados (*Node Store*) e para executar as instruções (*Functional Units*). A MMDM é uma máquina dinamica, permitindo a re-utilização e compartilhamento simultaneo de grafos por dados oriundos de várias instanciações através do mecanismo de coloração.

Um protótipo de um anel foi construído e esteve operacional durante vários anos, tendo sido descomissionado em 1991. A linguagem de programação para o protótipo e para os simuladores é SISAL.

2.1.2 DTN

O *DTN Dataflow Computer* é uma estação de trabalho gráfica comercialmente disponível, com 32 processadores a fluxo de dados estáticos. É construída pela companhia holandesa *Dataflow Technology Netherland* [24]. Esta estação usa processadores ImPP [21] desenvolvidos pela NEC, associados a um computador de duto VME Unix e a um subsistema gráfico com quatro processadores sistólicos de 64 MOPS.

2.2 Pesquisa na América do Norte

2.2.1 VIM

O grupo do prof. Dennis no MIT propos diversos dos conceitos básicos de fluxo de dados [6]. A pesquisa tem continuado desde 1968, sendo atualmente direcionada à construção de uma máquina de fluxo de dados estática de 1 GFLOP, direcionada para computação numérica intensa. A máquina VIM, mostrada na figura 2, é composta por um conjunto de redes de conexão, blocos de celas (CB), unidades funcionais (FU) e memórias de matriz (AM).

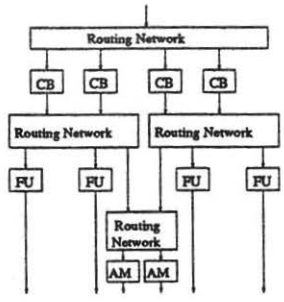

Figura 2: A arquitetura VIM

Os blocos de celas armazenam os grafos de dependência de dados e as estruturas de dados necessárias para implementar a regra de disparo, bem como os sinais de terminação típicos de fluxo de dados estático. Essas unidades também executam operações simples, tais como duplicação e eliminação de fichas. As unidades funcionais consomem os dados escalonados pelos blocos de celas. Os dados estruturados são armazenados nas memórias de matriz. As redes de conexão são tolerantes à latencia. O modelo de programação para a VIM é a linguagem VIMVAL, uma extensão da linguagem VAL que trata funções com objetos de primeira classe, que podem ser passados e retornados como parametros.

2.2.2 A Máquina MIT

Esta linha de pesquisas foi iniciada em 1975 em Irvine (Califórnia), sendo atualmente continuada no Massachussetts Institute of Technology (MIT) pelo grupo do prof. Arvind. Diversas arquiteturas a fluxo de dados foram propostas e estudadas, evoluindo para a proposta a MTTDA (*MIT Tagged-Token Dataflow Architecture*) [3], mostrada na figura 3:

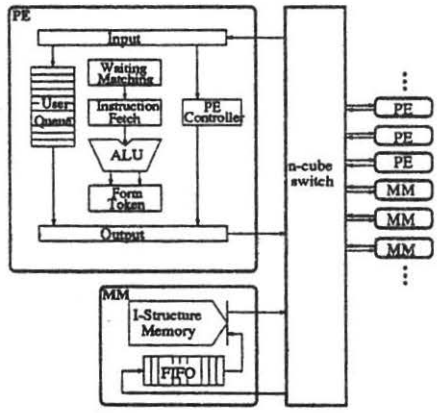

Figura 3: A arquitetura MTTDA

A MTTDA é uma máquina a fluxo de dados dinâmica e assíncrona, com 64 processadores interligados por uma rede de conexão n-cúbica, conectando também unidades de armazenamento denominadas *I-structure nodes*. Os nós *I-store* usam conceitos de coloração de fichas para permitir reutilização de grafos, armazenando dados de forma estática e permitindo leituras e escritas invertidas (leitura *antes* da escrita). Um protótipo com 256 placas, dirigido para computação simbólica e numérica e projetado para executar a 1 GIPS, está em construção. A linguagem de programação é a *Id*, uma linguagem específica para arquiteturas a fluxo de dados.

2.2.3 A Máquina Monsoon

A arquitetura da Monsoon [15] define uma máquina *multithreaded* de um endereço, incorporando o conceito de Memória de Fichas Explícito (ETS) [5]. O

computador Monsoon possui diversos processadores em *pipeline* e diversas memórias de *I-structure* interligados, como mostrado na figura 4.

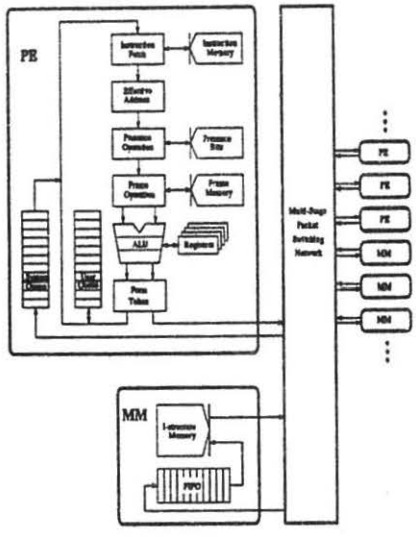

Figura 4: A arquitetura Monsoon

O conceito ETS requer a existência de ponteiro de quadro definindo o contexto e locais ao processador. Desta forma, nomes de ativação são restritos ao PE para o qual foram alocados, e essa instanciação é executada integralmente neste processador. Devido ao ETS, a regra de disparo é implementada como um *fetch* indexado e direto, dentro da porção de memória definida no quadro de referencias.

Há um conjunto de fichas completas (pares de dados e instruções) aguardando processamento implementado de forma distribuída: cada PE possui duas filas. A fila de alta prioridadade, chamada Fila de Sistema, obedece à disciplina FIFO. A fila de baixa prioridade, chamada Pilha do Usuário, obedece à disciplica LIFO. As unidades processadoras são compostas por um *pipeline* com os seguintes estágios: *Instruction Fetcher, Effective Address Calculator, Presence Bits Operator, Frame Operator, 3-stage ALU*, (incluindo *New Tag Calculator* e comunicando com o *Register Bank*), e *Token Formator*. Há uma via de dados para recirculação de fichas interna ao processador.

O protótipo de um PE unitário, de 4 MIPS, está operacional desde 1988, executando um núcleo de Sistema Operacional escrito em Id. Um projeto conjunto MIT/CalTech/Motorola está desenvolvendo uma placa com um PE de 10 MIPS/10 MFLOPS com interface VMS, implementado com 8 CI's CMOS de 10.000 portas [2]. A chave interprocessadores é implementada com CI's de chaveamento PaRC 4x4 [13]. Uma estação de trabalho Unix com um PE e executando Id deve ser lançada comercialmente pela Motorola ainda este ano. Um sistema com oito processadores a fluxo de dados, oito módulos de memória e quatro processadores também está planejada para ficar operacional em 1992.

2.2.4 Outras Arquiteturas

DDM1: A *Data Driven Machine* foi proposta pela Borroughs em 1975. É uma arquitetura a fluxo de dados, dinamico e sem o conceito de coloração de fichas, usando filas para distinguir entre instanciações. A topologia é uma árvore octal contendo elementos processadores. Cada elemento processador possui uma Fila de Agenda, uma Memória Atomica, um Processador Atomico e um Módulo de Chaveamento.

TDDP: O *Distributed Data Processor* foi projetado pela Texas Instruments em 1976 para investigar a praticidade de projetos a fluxo de dados estáticos de alta velocidade. A arquitetura do TDDP não exige a confirmação da execução da instrução [4], e um protótipo usando lógica TTL ficou operacional em 1978, usando quatro elementos processadores conectados em anel, executando ADA como linguagem de programação.

Epsilon-1, proposto pelo Sandia National Laboratories, é uma evolução do projeto DFAM, que foi um projeto investigativo de prococessador a fluxo de dados que permitiu a implementação de um sistema tão rápido quanto mini-supercomputadores da mesma época [9, 8]. O Epsilon usa a técnica de emparelhamento direto (*direct match*), que permite o acesso a dados armazenados na memória em apenas um ciclo de máquina e minimiza o *overhead* associado ao fan-out de instruções limitado, característico da maioria das implementações de máquinas a fluxo de dados.

Epsilon-2 é um processador sucessor do Epsilon - 1 que implementa o modelo dinâmico de fluxo de dados de forma completa. É uma máquina de granularidade variável, que usa um mecanismo híbrido de escalonamento de tarefas que permite a execução de código sequencial e/ou dirigido pela regra de disparo do dataflow. O Epsilon-2 possui diversos elementos processadores interconectados, como mostra a figura 5.

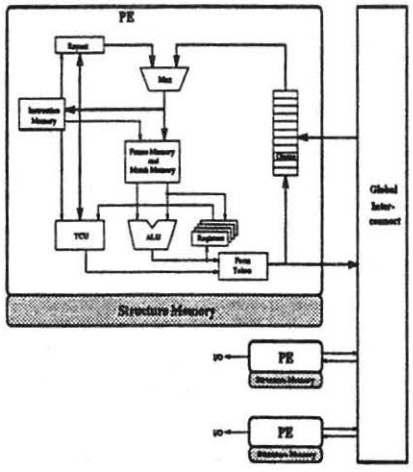

Figura 5: A arquitetura Epsilon-2

A arquitetura de cada elemento processador é inspirada no Epsilon-1 e no Monsoon, e pode executar código compilado a partir de Id, Sisal e Fortran. Os PE's permitem a conexão de *I-structures*, e a distribuição de carga é implementada alterando a imparcialidade da rede de comunicação, dificultando o tráfego na direção dos processadores mais sobrecarregados, num esquema reminiscente ao empregado no EM-4.

2.3 Pesquisa no Japão

2.3.1 SIGMA-1

Desenvolvido no *Eletrotechnical Laboratory* (ETL/MITI), o projeto SIGMA-1 é uma iniciativa governamental para estudar a praticidade da construção de um supercomputador a fluxo de dados dinamico para computação numérica [11]. Este projeto foi iniciado em 1982 e completado em 1988, com a construção de um computador com mais de 128 elementos processadores síncronos e com memórias de estrutras agrupadas em nós (*clusters*) interconectados por uma rede de conexão dual e hierárquica, como mostrado na figura 6.

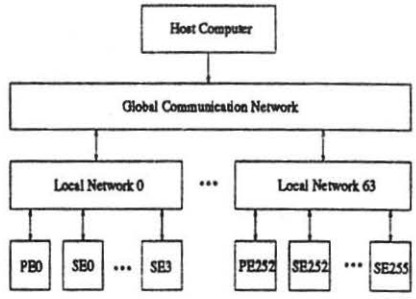

Figura 6: A arquitetura Sigma-1

O hardware é baseado em processadores CISC microprogramados, construídos com tecnologica CMOS LSI *semi-custom* com densidade de 10.000 portas/CI. Cada placa de elemento processador tem 27 circuitos integrados. O pico teórico de velocidade do computador com 128 nós é de 423 MFLOPS e 640 MIPS, e performances acima de 170 MFLOPS foram verificadas [19]. As instruções tem 40 bits para dados coloridos, e podem gerar resultados endereçados a até outras tres instruções. O SIGMA-1 executa a liguagem de programação DFC-2, um derivativo de C com restições de *single-assignment* e sem ponteiros.

2.3.2 EM-4

O EM-4 é a quarta geração de computadores dedicados para processamento simbólico do ETL/MITI. É uma máquina de granularidade variável empregando o conceito de arcos fortemente conectados [19]. Um processador integrado em um único CI RISC, denominado EMC-R, foi construído em 1988 com tecnologia *semi-custom* e densidade de 50.000 portas/CI. O EMC-R possui todas as funções básicas de uma arquitetura a fluxo de dados: chaveamento de pacotes, armazenamento de entrada (*buffering*); busca de instruções, emparelhamento de operandos (implementando a regra de disparo) e execução de instruções. Um protótipo de 80 nós está operacional desde 1990, com a estrutura mostrada na figura 7.

A velocidade de pico teórica do protótipo de 80

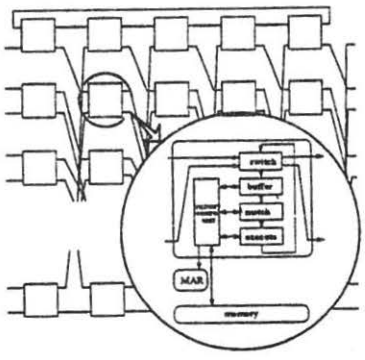

Figura 7: A arquitetura EM-4

nós é 1,1 GIPS e 14,6 GIPS para um protótipo de 1024 nós [19].

2.3.3 Outros Projetos no Japão

Diversas outras máquinas a fluxo de dados foram propostas, simuladas ou construídas, dentre as quais:

NEDIPS, desenvolvida na *Nippon Telegraph and Telephone Corporation (NTT)*, estatal japonesa de telecomunicações, no final da década de 1970. Possui uma arquitetura estática, dedicada a processamento de imagens e com extensões para permitir fichas múltiplas em cada arco. Implementada com CI's de baixa integração e circuitos discretos.

ImPP, denominado *Image Pipelined Processor*, também desenvolvido na estatal NTT no início dos anos 1980, é uma implementação integrada do NEDIPS num único circuito integrado, destinado a ser a célula básica para sistemas de alto desenpenho. O ImPP está comercialmente disponível desde 1985 [24].

EDDY, também desenvolvido pela NTT, é uma matriz de 16 nós com conectividade tipo "vizinho mais próximo" (*nearest neighbour*). Cada nó possui dois microprocessadores Z8000 e usa VALID como linguagem de programação. O projeto foi iniciado em 1980, o protótipo ficou operacional em 1983 e foi dado como encerrado em 1986 [20].

TopStar, da Universidade de Tóquio, é um protótipo de 16 processadores Z80, que explora fluxo de dados no nível procedural [18]. O objetivo do protótipo é o reconhecimento de caracteres impresso em Chines. O projeto foi iniciado em 1978 e terminou em 1982, sendo continuado pelo TopStar-II e subseqüentemente, pelo projeto PIE. A linguagem de programação dos protótipos é Lisp e Prolog.

DDDP, da *Oki Electric Industry Co.*, é um processador quádruplo usando lógica discreta e componentes de baixa integração, destinado a aplicações numéricas. A arquitetura é um processador a fluxo de dados convencional, tendo o projeto sido iniciado em 1980 e terminado em 1982 [14].

DFM, da NTT é um estudo para averiguar a adequação de processadores a fluxo de dados para manipulação simbólica e processamentos de listas com estratégia *lazy cons* [1]. O projeto foi iniciado em 1982 e um protótipo com dois processadores ficou operacional em 1985 quando foi iniciada a implementação de uma versão LSI em CMOS denominada DFM-II. A linguagem de programação é VALID.

PIM-D, da *Oki Electric Industry Co.*, é um protótipo com 16 processadores dedicados a aplicações numéricas, foi iniciado em 1982 e está operacional desde 1984, é financiado pelo projeto de quinta geração do governo japonês e usa Prolog como linguagem de programação.

EM-3, do ETL/MITI é uma máquina Lisp com 16 processadores, baseados no Motorola 68.000. O projeto foi iniciado em 1982, e o protótipo está operacional desde 1985 [25], usando EMLISP como linguagem de programação, um subconjunto de Lisp com restrições de atribuição única.

TIP, denominado *Template-based Image Processor* é um CI comercialmente disponível desenvolvido pela NEC para aplicações em processamento de imagens, usando uma arquitetura a fluxo de dados estática. O dispositivo está disponível desde a década de 1980 [21].

Q-v1, é um projeto conjunto da Universidade de Osaka, Sharp, Mitsubishi, Sanyo e Matsushita, dedicado para processamento de sinais e processamento de imagens [22]. Um protótipo com 8 processadores foi implementado usando uma família de cinco CI's desenvolvidos especialmente para esse fim. Os CI's usam a arquitetura de Manchester, com fluxo de dados dinamico baseado em coloração de fichas e empregando uma técnica de pipeline denominado *pipeline elástico*.

3 A arquitetura Wolf

3.1 Problemas Conhecidos

Muitas lições foram aprendidas durante a implementação da MMDM, a saber:

- A forma circular do pipeline aumenta o tempo de latencia e causa flutuações no tráfego das fichas.
- O número de estágios do pipeline da MMDM é maior que o necessário e atrapalha o desempenho.
- A implementação assíncrona dos estágios causa atrasos no pipeline e dificulta a depuração do protótipo.
- O comprimento fixo da palavra (32 bits) não é adequado, sendo muito pequeno para aplicações numéricas e excessivamente grande para aplicações simbólicas.
- A granularidade das instruções da MMDM aparenta ser mais fina que o desejável.
- A granularidade fixa da MMDM impossibilita a pesquisa sobre a granularidade ótima em processadores a fluxo de dados.
- Há redundância em diversos campos das fichas, aumentando de forma desnecessária o espaço de emparelhamento.

3.2 Um Exemplo Patológico

O grafo de dependência de dados da figura 8 mostra algumas das desvantagens de um código a fluxo de dados puro.

Nesse código, um ordenamento seqüencial das instruções é forçado pela dependencia dos dados. As arquiteturas von Neumann, devido à sua seqüencialidade implícita, são muito eficientes nesses casos. Num processador a fluxo de dados, nenhuma inferencia sobre o escalonamento da instrução Op2 pode ser efetuada no instante do escalonamento da instrução Op1. Entretanto é fácil ver que a regra de disparo que escalonou Op1 também disparou, devido à dependencia dos dados, as instruções Op2, Op3 e Op4. Como processadores a fluxo de dados tem dificuldades em identificar esse escalonamento implícito, a ficha com o resultado de Op1 precisa circular por todo o pipeline para disparar Op2, e assim sucessivamente para Op3 e Op4.

Essa ineficiencia é ainda maior se considerarmos que nenhuma instrução subseqüente a Op1 necessita de dados adicionais. A única entrada dessas outras instruções é o resultado de sua predecessora. Na terminologia da MMDM, essas fichas dirigidas a instruções unárias são denominadas *BYPASSES*. Essas fichas são, não obstante suas características, também enviadas para a unidade de emparelhamento, onde um mecanismo força o emparelhamento com um parceiro fantasma (do tipo Nulo). Simulações mostram que até 60% do tráfego na unidade de emparelhamento pode ser do tipo BYPASS.

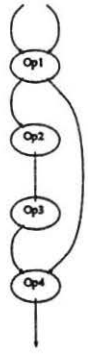

Figura 8: Código Patológico.

Outro problema detectado nessa situação é a criação de fichas de "vida curta", tais como o resultado intermediário de Op1, que é necessário apenas para disparar Op4. Num processador a fluxo de dados típico, esse resultado intermediário é armazenado na unidade de emparelhamento e quase imediatamente retirado. Essa ação desnecessária associada ao tráfego de fichas BYPASS, fazem com que a unidade de emparelhamento seja uma das mais lentas e críticas do pipeline.

3.3 Descrição Funcional

O proto-Wolf consiste das unidades interconectadas mostradas na figura 9. Na descrição que se segue a nomenclatura da MMDM foi mantida sempre que possível.

Uma ficha representa um dado no arco do grafo

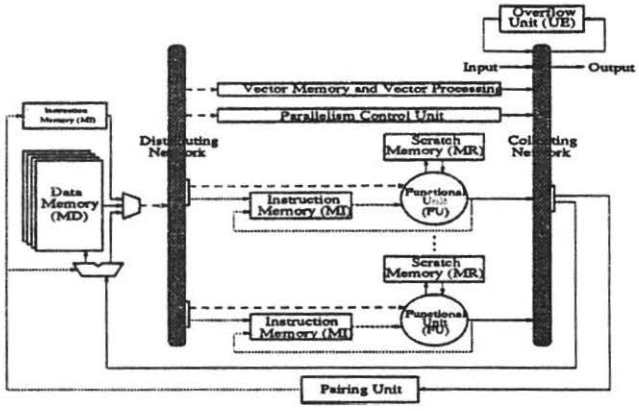

Figura 9: A Arquitetura a Fluxo de Dados Wolf

de dependencia sob interpretação. Fichas são inseridas na máquina pela via entrada da Chave de Distribuição. Da Chave de Coleta, as fichas passam para a Unidade de Emparelhamento se ela estiver disponível, ou são armazenadas na Fila de Fichas para distribuição posterior, caso a Unidade de Emparelhamento esteja ocupada. A Unidade de Emparelhamento é responsável pelo emparelhamento de operandos, combinando a saída da Memória de Dados para formar um pacote (*Group Package*) que é um par de fichas de mesma cor dirigidas para um mesmo nó. Esse pacote é distribuído pela Chave de Distribuição para uma das Memórias de instruções paralelas que esteja disponível. A Memória de Instruções armazena os detalhes do nó para o qual as fichas são dirigidas, e cria a partir do pacote, um pacote executável (*Executable Package*) com todas as informações necessárias para a execução da instrução enviando-o à sua Unidade Funcional associada. Os resultados produzidos pela unidade funcional podem ser do tipo *livre* ou *encadeado*. Resultados livres são passados para a Chave de Coleta recirculando no pipeline. Resultados encadeados permanecem nos registradores internos da Unidade Funcional para serem, num estágio posterior, usados como operandos de instruções subseqüentes.

Um dos detalhes do nó armazenados pela Memória de instruções é a informação de que um dado nó é parte de uma cadeia. Quando um pacote executável de uma cadeia é produzido, a Memória de instruções inicia por conta própria, um ciclo de busca usando o endereço de destino do pacote executável produzido como endereço de busca. Dessa forma, quando a Unidade Funcional termina o consumo do pacote executável anterior, um novo pacote executável está em condição de ser produzido pela Memória de Instruções. Esse novo pacote executável usará os resultados intermediários do pacote executável anterior como entrada.

Conectados entre as chaves de distribuição e coleta, estão uma ou mais unidades de Memória de Estruturas responsáveis pelo armazenamento de dados estruturados. A Memória de Estruturas aloca espaço sob demanda, e possui um mecanismo de contagem de referências que permite a liberação do espaço alocado automaticamente. As operações de leitura e escrita podem ser executadas em qualquer ordem, pois a Memória de Estruturas consegue armazenar internamente os pedidos de leitura para os dados ainda não produzidos.

3.4 Características da Arquitetura

3.4.1 Granularidade Variável

Uma característica marcante do Wolf é sua capacidade de executar código como um processador a fluxo de dados ou como um processador seqüencial, um conceito já mencionado por outros trabalhos em granularidade variável [12]. A arquitetura Wolf pode gerar uma cadeia de instruções para ser apresentada para uma Unidade Funcional onde as instru-

ções modificarão o estado da máquina descritos nos registradores internos como numa arquitetura von Neumann.

3.4.2 Localidade

Uma cadeia de instruções é gerada sempre que um diretivo for plantado pelo compilador. A deteção dessas cadeias no grafo de dependência é uma técnica conhecida, e não apresenta maiores dificuldades. Dessa mesma forma, fichas de vida curta podem ser facilmente detectadas e, através de instruções de armazenamento e busca plantadas pelo compilador, armazenadas nos registradores da Unidade Funcional para instruções seguintes. Os resultados intermediários das cadeias podem também ser direcionados para a Memória de instruções onde novas cadeias de instruções podem ser disparadas. Assim, diversas cadeias podem ser instanciadas em um único par Unidade Funcional - Memória de instruções antes que uma referência ao mecanismo de emparelhamento seja necessário.

3.4.3 Emparelhamento Direto

Devido à localidade forçada, a geração excessiva de cores para distinguir contextos é desestimulada. A implementação da unidade de *Throttle* [16] no mesmo nível hierárquico das unidades funcionais facilita a reciclagem de nomes de ativação (cores), e o emprego da Unidade de Processamento Vetorial dispensa a maioria dos índices nas fichas. Esses fatores concorrem para uma acentuada diminuição no espaço de emparelhamento, que agora fica compatível com os espaços de endereçamento usados em sistema de memória virtual. A Memória de Dados pode dessa forma ser gerenciada usando estas técnicas já provadas. A Memória de Dados no Wolf, é uma memória virtual paginada endereçada pela Unidade de Emparelhamento.

A Chave de Coleta tem duas vias separadas para a Unidade de Emparelhamento: uma leva fichas para a Memória de Dados a outra leva endereços para a Unidade de Emparelhamento. A Unidade de Emparelhamento usa esse endereço para verificar a presença do par. Se não há um par, um sinal de escrita é gerado e o dado apresentado é armazenado. Se há um par, um sinal de leitura é gerado e a Memória de Dados apresenta o dado endereçado.

4 Detalhes de Implementação e Alvos

- Comprimento de palavra variável, 32 a 96 bits.
- Tamanho de ficha reduzido (comparado à MMDM): 26, 56, 152 e 164 bits.
- Granularidade variável: fina, cadeia de instruções e grossa.
- Unidades Funcionais implementadas com técnicas RISC e memória local de registradores.
- Unidade de Emparelhamento independente da Memória de Dados. Endereçamento direto com memória virtual.
- Emparelhamento de até quatro fichas.
- Memórias de instruções acopladas às unidades funcionais, capazes de gerar cadeias de instruções

As unidade do Wolf trocam informações exclusivamente através de fichas que são divididas em campos. O comprimento das fichas é um dos parametros críticos da implementação e estão tentativamente definidos como mostra a tabela 1.

5 Programa Simulador

Um simulador escrito em C++ (versão 2.0 da AT&T) executado em estações SUN está operacional. Escolhemos a linguagem C++ pelas facilidades oferecidas pela programação orientada a objeto. Cada módulo do Wolf é definido como uma classe, facilitando replicação, independencia de efeitos colaterais, re-utilização de código e modularização. Outra característica extensivamente explorada foi a sobrecarga de operadores e funções.

A simulação da memória de dados, fila de fichas e memória de instruções foi feita usando matrizes de forma a permitir um acesso direto à posição desejada sem usar técnicas de *hash*. Dessa forma tivemos que levar em consideração as limitações da máquina hospedeira como velocidade de processamento e quantidade de memória disponível. Para

Campo	Nome	Função	Tamanho
$\mathcal{A}$	Ativação	Identificação de ativação	10
$\mathcal{C}$	CodOp	Código de Operação	6
$\mathcal{D}$	Dados	Dados	32
$\mathcal{E}$	Emparelhamento	Número de fichas emparelhadas	2
$\mathcal{N}$	Nó	Endereço da instrução no programa	12
$\mathcal{T}$	Tipo	Tipo do dado	4

Tabela 1: Campos das Fichas

economia de memória tivemos que usar uma técnica chamada *campo de bits* que consiste em fazer uma pré-avaliação das variáveis e em seguida limitar a quantidade de bits que cada variável poderá conter limitando assim o número máximo que ela poderá representar. A quantidade de memória necessária para a execução do programa foi assim bastante reduzida.

O simulador possui sete classes cada uma representando uma parte da arquitetura: **E_S** para a entrada e saída, **MI** para as memórias de instruções, **P** para as unidades funcionais, **CC** para a chave de coleta, **T_Q** para a fila de fichas, **C_D** para a chave de distribuição e finalmente **MD** para a memória de dados.

5.1 Entrada e Saída

O simulador executa código no padrão COD gerado pelo compilador Sisal e embute um interpretador da linguagem de máquina da MMDM. O módulo de entrada e saída possui três funções privadas à classe que interpretam as informações lidas dos arquivos. As funções são: *conversão* que analiza e converte as instruções, *converte_dado* que analiza e converte os dados e *acha_c_oper* que converte o código de operação em uma seqüencia de caracteres pré-estabelecida.

5.2 Memórias de Instruções

Para armazenar as fichas de instruções usamos uma matriz unidimensional que simula uma memória de acesso direto. Essa matriz armazena uma estrutura de dados com cinco campos.

5.3 Unidades Funcionais

Na implementação das duas unidades funcionais foi necessária também a criação de uma variável inicializada na definição do objeto para identificarmos qual dos dois processadores estava sendo usado em determinado tempo. Nas unidades funcionais estão definidas as operações aritméticas como adição e multiplicação e algumas operações de desvio condicional que o WOLF pode realizar.

5.4 Chave de Coleta

A Chave de Coleta faz a seleção de entradas e saídas possuindo um *buffer* interno que armazena as fichas das entradas em ordem crescente do campo *tempo*. Isso diminui bastante o número de interações necessárias para selecionar qual entrada irá para qual saída.

5.5 Fila de Fichas

A fila de fichas é implementada usando uma matriz unidimensional para simular uma lista encadeada. Os ponteiros que controlam as posições livres e as ocupadas são definidos como variáveis locais à classe. Implementamos também um mecanismo que detecta o *overflow* da lista.

5.6 Chave de Distribuição

A chave de distribuição é a classe cujo código é o mais simples. Possui apenas tres laços de *if's* cuja função é o de selecionar por qual anél a ficha deverá ser mandada.

5.7 Memória de Dados

A memória de dados é implementada usando uma matriz tridimensional que simula uma memória de acesso direto. A classe possui quatro funções privadas que ajudam no emparelhamento das fichas. Essa estrutura é responsável pela maior parte da memória alocada pelo simulador.

6 Resultados Obtidos com a Simulação

Analisamos o simulador WOLF sob o aspecto de ocupação da fila de fichas e ocupação da memória

de dados para validá-lo e para comparar o comportamento da arquitetura Wolf. Adotamos dois *benchmarks* já extensivamente usados nesse contexto: cálculo da função de Ackermann e integração binária usando a regra do trapézio.

6.0.1 Função de Ackermann

A função de Ackermann é usada em teoria de funções recursivas para discriminar entre arranjos de funções computáveis, pois pode ser formalmente mostrado que ela não está contida no conjunto de funções recursivas primitivas. Essa função é freqüentemente usada para testar arquiteturas paralelas e a eficiencia de chamadas a procedimento em implementação de linguagens. Essa função possui as características desejáveis de um algoritmo curto e recursivo que não gera valores inteiros muito elevados.

O número de chamadas ao algoritmo para o cálculo de Ackermann(3,n) é:

$$\frac{128*4^n-120*2^n+9*n+37}{3}$$

O resultado pode ser obtido por:

$$\text{Ackermann(3,n)} - 1 = 2^{n+3} - 4$$

O simulador calculou corretamente os resultados para vários valores da função de Ackermann.

6.0.2 Integração Binária

A função a ser integrada é: $y = x^2 - 6x - 10$, integrada entre os parâmetros de entrada **L** e **R**. Essa rotina é uma rotina com alto grau de recursividade que cria uma árvore de processos com 2^{n-1} processos (n é um parâmetro de entrada). O grau de paralelismo também é muito alto pois não há dependencia nos dados entre os vários subintervalos e dessa forma a regra do trapézio pode ser aplicada em paralelo em todos eles.

O simulador forneceu respostas corretas para o valor da integral para vários domínios dessa função.

7 Resultados

Os gráficos das figuras 10, 11 e 12 mostram resultados preliminares do comportamento das unidades internas do Wolf.

A figura 10 mostra um comportamento oscilatório da ocupação de memória no cálculo da função de

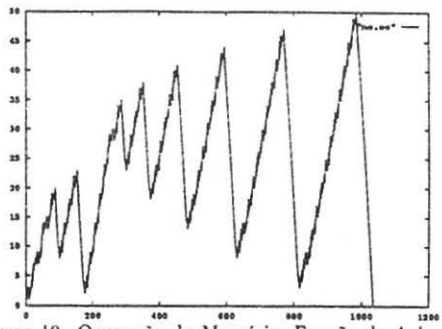

Figura 10: Ocupação da Memória, Função de Ackermann

Ackermann compatível com os resultados já publicados [17]. A figura 11 mostra um comportamento da fila de fichas compatível com o conhecido e publicado para o algoritmo de integração binária.

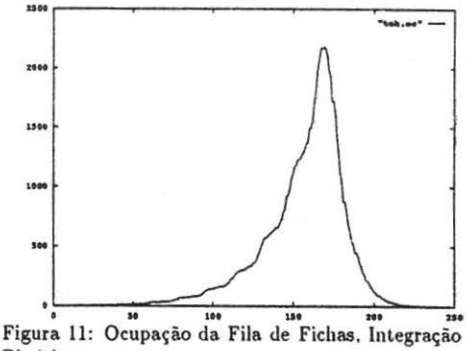

Figura 11: Ocupação da Fila de Fichas, Integração Binária

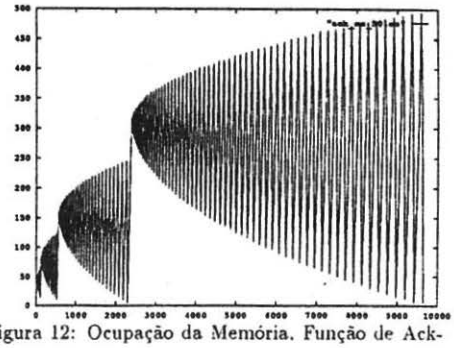

Figura 12: Ocupação da Memória, Função de Ackermann

A figura 12 mostra um comportamento surpreendente para a ocupação de memória na função de Ack-

ermann (sob diferentes condições), extremamente diferente do esperado e conhecido. Este comportamento pode ser uma característica da arquitetura Wolf ou um efeito de *aliasing* na implementação do simulador. A investigação nesse sentido prossegue.

Referências

[1] M. Amamiya and R. Hasegawa. Dataflow Computing and Eager and Lazy Evaluation. *J. New Generation Computers*, 2(8):105–129, 1984.

[2] Arvind, M. L. Dertouzos, R. S. Nikhil, and G. M. Papadopoulos. PROJECT DATAFLOW, a Parallel Computing System Based on the Monsoon Architecture and the Id Programming Language. Technical Report CSG Memo 285, LCS, MIT, 1988.

[3] Arvind, V. Kathail, and K. Pingaley. A Dataflow Architecture With Tagged Tokens. CSG Memo 174, LCS, MIT, Cambridge, Mass., USA, Sep 1980.

[4] M. Corsish, D. W. Wogan, and J. C. Jensen. The Texas Instruments Distributed Processor. In *Proc. Louisiana Computer Exposition*, 189–193, Lafayette, LA, USA, Mar 1979.

[5] D. E. Culler and G. M. Papadopoulos. The Explicit Token Store. *J. Paralle and Distributed Computing*, 10(4), Jan 1991.

[6] J. B. Dennis. Data Flow Computation. In M. Broy, editor, *NATO ASI Series, v.F14, Control Flow and Data Flow: Concepts of Distributed Programming*, 346–397. Springer-Verlag, Berlin, 1985.

[7] J. F. Foley. Manchester Dataflow Machine: Benchmark Test Evaluation Report. Internal Report UMCS–89–11–1, DCS, Univ. Manchester, England, Nov 1989.

[8] V. G. Grafe and J. E. Hoch. Implementation of the Epsilon Dataflow Processor. In *Proc. 23rd Hawaii Int. Conf. on System Sciences*, 1990.

[9] V. G. Grafe and J. E. Hoch. The Epsilon-2 Multiprocessor System. *J. Parallel and Distributed Computing*, 10(4):309–318, Dec 1990.

[10] J. R. Gurd, I. Watson, and J. R. W. Glauert. A Multilayered Dataflow Computer Architecture. Internal report, DCS, Univ. Manchester, Mar 1980.

[11] K. Hiraki, T. Shimada, and K. Nishida. A Hardware Design of the SIGMA-1 — A Data Flow Computer for Scientific Computations. In *Proc. Int. Conf. on Parallel Processing*, 524–531. IEEE, 1984.

[12] R. A. Iannucci. A Dataflow/von Neumann Hybrid Architecture. Technical Report TR–418, LCS, MIT, Cambridge, Mass., USA, 1988.

[13] C. F. Joerg. Design and Implementation of a Packet Switched Routing Chip. Masters thesis, Dept. Electrical Eng. and Computer Science, MIT, Cambridge, Mass., USA, 1988.

[14] M. Kishi, H. Yasuhara, and Y. Kawamura. DDDP: A Distributed Data Driven Processor. In *Proc. 10th Annual Int. Symp. Computer Architecture*, 236–242. IEEE, 1983.

[15] G. M. Papadopoulos. Implementation of a General Purpose Dataflow Multiprocessor. PhD Thesis, Technical Report TR 432, LCS, MIT, Cambridge, Mass., USA, Sep 1988.

[16] C. A. Ruggiero. Throttle Mechanisms for the Manchester Dataflow Machine. Technical Report Series UMCS-87-8-1, DCS, Univ. Manchester, England, Jul 1987.

[17] A. Garcia Neto Distributed Simulation Using "Relaxed Timing" PhD Thesis, DCS, Univ. Manchester, England, 1991.

[18] T. Suzuki, K. Kurihara, H. Tanaka, and T. Moto-oka. Procedure Level Data Flow Processing on Dynamic Structure Multimicroprocessors. *J. Information Processing*, 1(5):11–16, 1982.

[19] T. Shimada T. Yuba et al. Dataflow Computer Development in Japan. *ACM*, 140–147, 1990.

[20] N. Takahashi and M. Amamiya. A Data Flow Processor Array System — Design and Analysis . In *Proc. IEEE 10th Annual Int. Symp. Computer Architecture*, 243–250. IEEE, 1983.

[21] T. Temma, S. Hasegawa, and S. Hanaki. Dataflow Processor for Image Processing. *Proc. on Mini and Microcomputers*, 5(3):52–56, 1980.

[22] H Terada, H Nisikawa nd K. Asada, S. Matsumoto, S. Miyata, S. Komori, and K. Shima. VLSI Design of a One-Chip Data-Driven Processor: Q-v1. In *Proc. Fall Joint Computer Conf.*. ACM/IEEE, 1987.

[23] F. J. Valente. Estudo de arquiteturas risc. Master's thesis, DFCM, IFQSC, Universidade de S ao Paulo, 1991.

[24] A. H. Veen and R. van den Born. The RC Compiler for the DTN Dataflow Computer. *J. Parallel and Distributed Computing*, 10(4):319–332, Dec 1990.

[25] Y. Yamagushi, K. Toda, J. Herath, and T. Yuba. EM-3: A LISP-Based Data-Driven Machine. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, 524–532. ICOT, 1984.