Mecanismo de Busca Especulativa de Múltiplos Fluxos de Instruções[1]

Rafael R. dos Santos[2]
Philippe O. A. Navaux[3]

Universidade Federal do Rio Grande do Sul
Curso de Pós-Graduação em Ciência da Computação - CPGCC
Av. Bento Goncalves, 9500 Cx.P.15064 - CEP:91501-970
Porto Alegre - RS - Brasil

Resumo

Este trabalho apresenta um novo modelo de busca especulativa de múltiplos fluxos de instruções em arquiteturas superescalares. A avaliação de desempenho de uma arquitetura superescalar com esta característica é também apresentada como forma de validar o modelo proposto e comparar seu desempenho frente a uma arquitetura superescalar real. O modelo em questão pretende eliminar a latência de busca de instruções introduzida pela ocorrência de comandos de desvio em pipelines superescalares. Algumas considerações sobre o modelo descrito são apresentadas ao final do trabalho assim como sugestões para trabalhos futuros.

Palavras Chave: Paralelismo de Baixo Nível, Arquiteturas Superescalares, Arquiteturas Multifluxo

Abstract

This work presents a new model for multistream speculative instruction fetch in superscalar architectures. Also, the performance evaluation of a superscalar architecture with this feature is presented in order to validate the model and to compare its performance with a real superscalar architecture. This model intends to eliminate the instruction fetch latency introduced by branch instructions in superscalar pipelines. Finally, some considerations about the model are presented as well as suggestions to future works.

Keywords: Instructions-Level Parallelism, Superscalar Architetcures, Multistreamed Architectures

[1]Projeto finaciado parcilamente pelo ProTem-CC fase III e University Research & Development Grant Program - Cray Research Inc.

[2]B.Sc.; Mestrando – E-mail: rrsantos@inf.ufrgs.br

[3]Prof. Dr.; Inst. Informática – E-mail: navaux@inf.ufrgs.br

1 Introdução

Arquiteturas superescalares potencialmente são capazes de executar mais de uma instrução em um único ciclo de máquina. Dependências de controle possuem um papel determinante no desempenho destas e são provocadas pelos comandos de desvio. A instrução que deve ser executada após uma instrução de desvio só é conhecida quando a instrução de desvio é completada. Cria-se assim uma relação de dependência, que obriga a execução seqüencial da instrução de transferência e da instrução seguinte. A conseqüência é uma redução do número médio de instruções despachadas por ciclo, e do desempenho da arquitetura [5, 6, 8].

Arquiteturas superescalares empregam técnicas de previsão de desvios juntamente com a execução especulativa de instruções para contornarem o problema das dependências de controle ou diminuirem o número de desvios que causam penalidade no desempenho do processador. Estas técnicas até então foram utilizadas como a forma mais eficiente e econômica para contornar o problema dos desvios. O aumento na capacidade de integração e a redução no custo do hardware está possibilitando que novas técnicas, anteriormente inviabilizadas em função de seu custo, sejam consideradas como técnicas mais agressivas de exploração do paralelismo de instrução.

É essencial observar que a previsão de desvios possibilita apenas a continuidade da busca de instruções na presença de dependências de controle. Estes mecanismos não atacam a outra parte do problema, qual seja, o fato das dependências de controle tornarem a execução dependente do resultado de um desvio condicional [2]. Uma instrução de desvio condicional deve ser antes executada para que seja determinado se as instruções acessadas antecipadamente pelo mecanismo de previsão podem, de fato, serem executadas.

Muitas pesquisas têm sido desenvolvidas com o objetivo de encontrar novos paradigmas que sustentem a ordem de oito a dezesseis instruções executadas por ciclo [15]. Embora as arquiteturas superescalares possam executar mais de uma instrução por ciclo, deparam-se com sérios problemas na busca de tais índices.

Uma alternativa para a previsão de desvios é executar especulativamente ambas as ramificações do desvio, ao invés de prever se o desvio é tomado ou não-tomado, e anular a ramificação incorreta tão logo o resultado do desvio seja conhecido. Desta forma, a penalidade do desvio pode ser eliminada, embora sejam necessários mais recursos computacionais [8]. Com o aumento da densidade de integração e o barateamento do hardware esta alternativa compõe uma forte tendência.

Neste trabalho é analisado o desempenho de uma arquitetura superescalar capaz de acessar instruções pertencentes a ^{a}uxos (*Paths*) distintos aumentando o número de instruções que são disponibilizadas para despacho. O estágio de busca (*fetch*) foi modificado para acessar e armazenar estas instruções conforme as instruções de desvio são encontradas no programa que está sendo executado.

O modelo de busca especulativa de instruções, foi avaliado como uma alternativa que permite o encadeamento de instruções pertencentes a ^{a}uxos distintos evitando que a fila de instruções seja esvaziada e aumentando o número de instruções disponíveis ao escalonamento. Com esta filosofia pretende-se aumentar o *IPC* de arquiteturas superescalares

possibilitando *speed-ups* mais elevados e próximos ao ideal.

Na seção 2 são discutidas as arquiteturas superescalares. O ambiente experimental é descrito na seção 3. Na seção 4 são mostrados os fatores responsáveis pelo esvazimento da fila de instruções em uma arquitetura superescalar real. Seu desempenho é comparado com o desempenho de uma arquitetura ideal que não apresenta ciclos adicionais ocasionados pela previsão de desvios tomados, contendo mecanismo perfeito de previsão de desvios. Na seção 5 é definido o mecanismo de busca especulativa em múltiplos "uxos utilizado nos experimentos. O desempenho deste mecanismo é analisado na seção 6 seguido pelas conclusões finais apresentadas na seção 7.

2 Arquiteturas Superescalares

Nesta seção apresenta-se a descrição do funcionamento de alguns estágios do pipeline tipicamente encontrado em arquiteturas superescalares. A figura 1 mostra a anatomia típica de uma arquitetura superescalar.

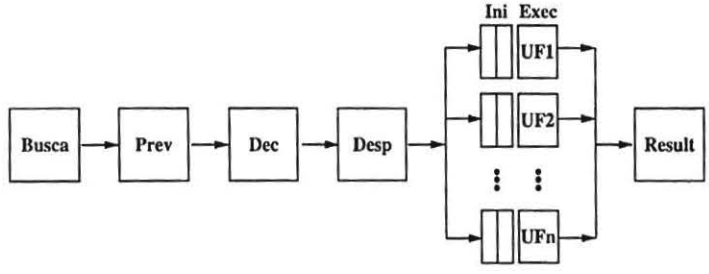

Figura 1: Anatomia Típica de uma Arquitetura Superescalar

No ciclo n, o estágio de *Busca* acessa "largura de busca" instruções (*Fetchwidth*) na cache de instruções colocando-as num buffer denominado buffer de busca (*Fetch Buffer*).

No ciclo $n+1$, o estágio de *Previsão* transfere estas instruções, do buffer de busca para a fila de instruções (*Instruction Queue*), examinando o tipo de cada uma delas. Quando uma instrução de desvio é encontrada, uma previsão do destino provavelmente alvejado por esta instrução é feita. A previsão feita define o caminho que será seguido pelo estágio de *Busca* no próximo ciclo estabelecendo uma dependência de controle.

Uma dependência de controle existe de um comando Ci para Cj se o comando Cj deve ser executado somente se o comando Ci produzir certo valor [7]. Este tipo de dependência ocorre, por exemplo, quando o comando Ci é um comando condicional e Cj é executado somente se a condição avaliada por Ci for verdadeira.

Um dos maiores problemas no projeto de *pipelines* é garantir o "uxo contínuo de instruções através do *pipeline* permitindo desta forma aproximação ao desempenho máximo teórico. O "uxo de instruções pode ser interrompido por dois motivos. Primeiro, o tempo de acesso à memória para buscar uma instrução é tão longo que uma requisição, feita pelo

estágio de busca, por outra instrução, não é satisfeita no tempo de estágio do *pipeline*. Segundo, a troca no "uxo esperado de instruções, devido a um desvio por exemplo, faz com que parte do conteúdo do *pipeline* seja descartado, e o *pipeline* seja recarregado.

Um desvio é tomado (*taken*) sempre que o "uxo de controle é desviado para o destino especificado como endereço alvo do desvio (*target address*). Desvios não-tomados (*not-taken*) são desvios que não transferem o "uxo de controle para o endereço alvo e portanto prosseguem a execução através da instrução adjacente à instrução de desvio. Tipicamente, desvios condicionais ou de controle de *loops* podem ser tomados ou não. Desvios incondicionais são sempre tomados.

Quando um desvio é previsto como tomado, todas as instruções acessadas antecipadamente pelo estágio de busca, que se encontram após a instrução de desvio, são descartadas. O endereço alvo do desvio é informado ao estágio de *Busca* que redireciona o acesso à cache de instruções recomeçando a busca através do caminho previsto. Em caso contrário, ou seja, quando o desvio é previsto como não-tomado, o estágio de previsão simplesmente continua a transferir as instruções para a fila de instruções até que o buffer de busca esteja vazio ou a fila de instruções esteja cheia, por exemplo.

O terceiro estágio faz a decodificação das instruções presentes na fila de instruções no ciclo $n + 2$. Estas instruções são despachados pelo estágio de *Despacho* no ciclo $n + 3$, alocando entradas nas estações de reserva das unidades funcionais existentes na arquitetura e no buffer de reordenação. No ciclo seguinte, $n+4$, instruções que estão prontas são enviadas às unidades funcionais livres que as executarão no ciclo $n + 5$, conforme as dependências de dados forem resolvidas. Após serem executadas, as instruções são reordenadas no ciclo $n + 6$ e os registradores da máquina são atualizados com os valores produzidos por instruções executadas corretamente.

Descreveu-se de maneira sucinta o funcionamento de sete estágios comumentemente encontrados em arquiteturas superescalares. Descrições mais detalhadas poderão ser encontradas em [10, 11, 4, 16].

3 Ambiente Experimental

O presente trabalho tem como objetivo principal avaliar o desempenho de uma arquitetura superescalar multi"uxo. Com o intuito de fornecer resultados e dados consistentes foram realizadas centenas de simulações utilizando-se 4 (quatro) simuladores e 4 (quatro) benchmarks do conjunto SPECint95 [17].

Os experimentos realizados visam mostrar que o desempenho de arquiteturas superescalares ainda está muito abaixo do desempenho ideal devido a uma série de fatores que serão mostrados na seção 4. Após mostrar quais são estes fatores, apresenta-se um mecanismo de busca especulativa de instruções em múltiplos "uxos capaz de reduzir alguns destes fatores e aumentar o potencial de paralelismo da arquitetura superescalar multi"uxo.

Os simuladores de 3 arquiteturas superescalares empregados são do tipo *trace-driven* que utilizam arquivos de *trace* gerados por um simulador da arquitetura SPARC [13]. Abaixo descreve-se o funcionamento de cada arquitetura superescalar empregada nos

experimentos.

1. Simulador Ideal
   Simula a execução de programas fazendo previsão de desvios perfeita. No estágio de *Previsão* o simulador lê os arquivos de *trace* e passa a buscar instruções no "uxo correto de execução do programa eliminando a latência de acesso normalmente causada pela ocorrência de instruções de desvio.

2. Simulador Real
   Simula a execução de programas fazendo previsão de desvios em dois níveis [9]. Nestas simulações verifica-se as causas principais do esvazimento da fila de instruções em função da ocorrência de desvios e a conseqüente queda de desempenho.

3. Simulador Mul"ux
   Simula a execução de programas utilizando a técnica de busca especulativa em múltiplos "uxos. Nestas simulações verifica-se a redução drástica do esvaziamento da fila de instruções e um aumento potencial no paralelismo disponível. Além destas análises são apresentadas outras considerações essenciais para futuras implementações.

Foram empregados nos experimentos os programas *compress, go, ijpeg* e *li* todos pertencentes ao conjunto SPECint95. Todas as simulações realizadas cobriram a execução de *10.000.000* de instruções para cada programa de teste.

Tabela 1 Características dos Benchmarks

Benchmarks	Instruções Simuladas	Ciclos	CPI	Desvios Condicionais	(%)	Tomados (%)	Não-Tomados (%)
compress	10×10^6	12717265	1.27	1055630	11	82.92	17.08
go	10×10^6	12572511	1.26	1070617	11	70.99	29.01
ijpeg	10×10^6	12656689	1.27	2301463	23	80.82	19.18
li	10×10^6	12982557	1.30	2023238	20	74.00	26.00

Os dados apresentados na tabela 1 foram extraídos a partir da execução seqüencial de cada um dos *benchmarks*. Esta execução foi realizada a partir do simulador *SIMSPARC - execution-driven* da arquitetura SPARC, gerando os arquivos de *traces* que foram utilizados nas demais simulações.

4 Análise do Problema

Nesta seção são comparados o desempenho (*speed-up*) de uma arquitetura que apresenta mecanismo perfeito de previsão de desvios com o desempenho de uma arquitetura real que possui mecanismo de previsão de desvios e execução especulativa existentes em microprocessadores atualmente comercializados.

Os resultados apresentados nesta seção foram obtidos com configurações idênticas para as arquiteturas ideal e real de acordo com os seguintes parâmetros:

- Largura de Busca de 8 instruções;
- Buffer e Fila de Instruções com 16 e 32 posições, respectivamente;
- *Brach History Table*: 1024 entradas/*4-way set associative*;
- *Pattern History Table*: 4096 entradas;
- Largura de Despacho de 8 instruções;
- Unidades Funcionais (homogêneas) variando de 2 a 8, cada uma com 8 estações de reserva;
- 1 Unidade de Desvios com 8 estações de reserva;
- 8 Barramentos de Resultados, e;
- Buffer de Reordenação com 64 posições.

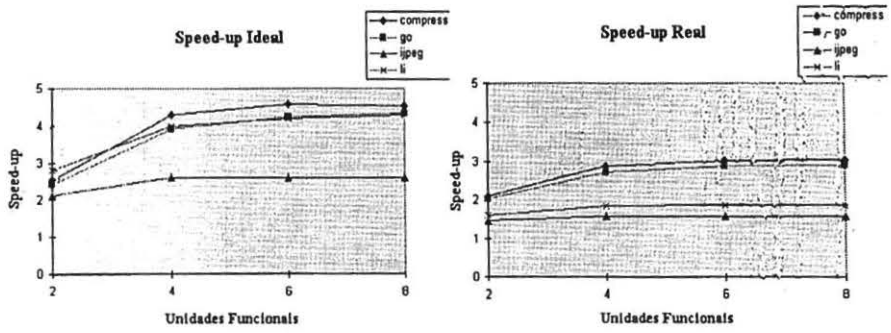

Figura 2: Speed-ups das arquiteturas ideal e real

Os gráficos da figura 2 mostram que o desempenho da arquitetura superescalar real é bastante inferior ao desempenho da arquitetura ideal. Segundo [3], os fatores que limitam o desempenho da arquitetura ideal são as dependências de dados e a disponibilidade de recursos. Já na arquitetura real existe um outro fator predominante que causa o esvaziamento da fila de instruções e a redução do número médio de instruções despachadas para execução. O número de ciclos com despacho nulo é bastante superior na arquitetura real em função da descontinuidade na transferência de instruções para a fila de instruções decorrente da previsão de desvios tomados.

O gráfico da figura 3 (*Ciclos com Despacho Nulo*) mostra que a existência de ciclos com despacho nulo na arquitetura real deve-se a três fatores: disponibilidade de recursos, fila de instruções vazia e despacho bloqueado devido a previsões incorretas.

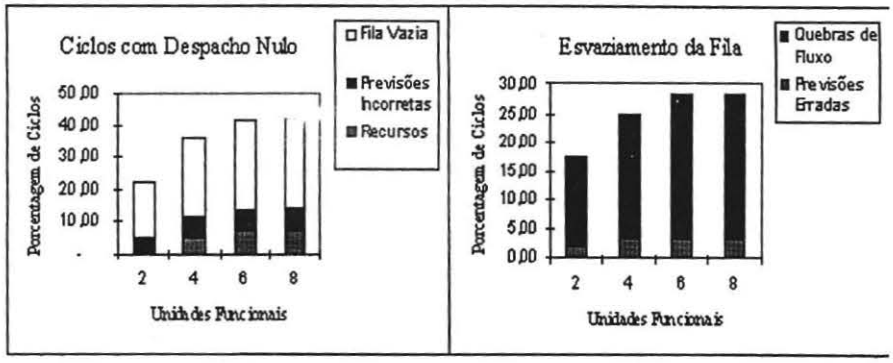

Figura 3: Gráfico de Ciclos com Despacho Nulo e Ocorrência de Fila Vazia

Como se pode observar, a existência de ciclos com despacho nulo deve-se principalmente a ocorrência de fila de instruções vazia. É importante salientar que o aumento da taxa de indisponibilidade de recursos, mesmo com o aumento do número de unidades funcionais, ocorre devido a profundidade de especulação mantida em todas as configurações.

Analisando-se os fatores que causam o esvaziamento da fila de instruções pode-se notar que as constantes quebras de "uxo são o principal fator. No gráfico 3 (*Esvaziamento da Fila*) verifica-se que a fila de instruções é esvaziada devido as previsões incorretas e as constantes quebras de "uxo. No entanto, as constantes quebras de "uxo são a principal causa do esvaziamento da fila de instruções.

Na seção 5 apresenta-se uma técnica capaz de reduzir o esvaziamento da fila de instruções na ocorrência de desvios. Resultados de simulações realizadas são apresentados e outros aspectos são analizados.

5 Busca Especulativa de Múltiplos Fluxos de Instruções

Em [14], é apresentada uma técnica para a redução do problema dos desvios condicionais, denominada *Fetch Taken and not-Taken Paths*. Em [3], é apresentado um novo modelo de execução paralela de instruções baseado na técnica mencionada. Neste novo modelo, ambas as ramificações de um desvio são buscadas antecipadamente e armazenadas no *buffer* de busca.

A previsão de desvios é ainda empregada neste modelo. Quando a previsão é feita, os dois possíveis caminhos do desvio, já estão presentes no *pipeline*. Desta forma, não se faz necessário redirecionar o estágio de busca ao caminho previsto, haja visto que as instruções pertencentes a este já se encontram no *pipeline*.

É desejável que a cada ciclo, toda a largura de busca seja preenchida com instruções, e que estas sejam transferidas para a fila de instruções, possibilitando a decodificação e posteriormente o despacho para execução, mesmo quando um desvio é previsto como tomado. Além disto, novas instruções podem ser colocadas no *buffer* de busca no ciclo

imediatamente após uma previsão. Aliado a um mecanismo de previsão que ofereça altas taxas de acerto, este funcionamento pode resultar em um "uxo elevado de entrada na fila de instruções mascarando as quebras de "uxo.

Para realizar esta operação, a arquitetura deve detectar uma instrução de desvio no próprio estágio de busca e, no ciclo seguinte, iniciar a busca de instruções nos dois possíveis caminhos do desvio. Quando um desvio é previsto como tomado, a transferência para a fila de instruções não é interrompida após a instrução de desvio.

Nas máquinas superescalares convencionais (instruções de um único "uxo de controle) quando um desvio é previsto como tomado, a busca é redirecionada para o caminho não adjacente à instrução de desvio (quebra de "uxo). Neste caso, são introduzidos alguns ciclos até que o endereço de busca tenha sido corrigido e a instrução sucessora, no "uxo lógico, tenha sido acessada (latência de acesso). O número de ciclos desperdiçados depende da arquitetura em questão.

Na técnica acima, três fatores devem ser considerados:

- Taxa de acertos da previsão de desvios: o mecanismo de previsão de desvios deve garantir uma taxa alta de acertos, para que o encadeamento de instruções mantenha a continuidade da entrada de instruções;
- Latência de acesso às instruções: instruções no destino do desvio devem ser acessadas a tempo de serem encadeadas. O acesso das instruções no destino do desvio pode provocar falhas na memória cache;
- Possibilidade da criação de novos "uxos quando um novo desvio é encontrado.

5.1 O Novo Estágio de Busca

Na arquitetura superescalar multi"uxo estudada o estágio de *Busca* foi modificado para permitir que ambos os caminhos de um desvio fossem acessados e armazenados após a previsão de um desvio ter sido feita.

A figura 4 sugere a nova estrutura de buffers do estágio de *Busca* do pipeline multi"uxo. O número de buffers determina a profundidade de especulação na busca de instruções.

Cada "uxo possui quatro elementos independentes:

- Program Counter
- Status Bit
- Children List
- Fetch Buffer

No *Fetch Buffer* são armazenadas as instruções acessadas. O *PC* é empregado para apontar as instruções que estão ao longo de um "uxo. O *status bit* indica a ocupação da

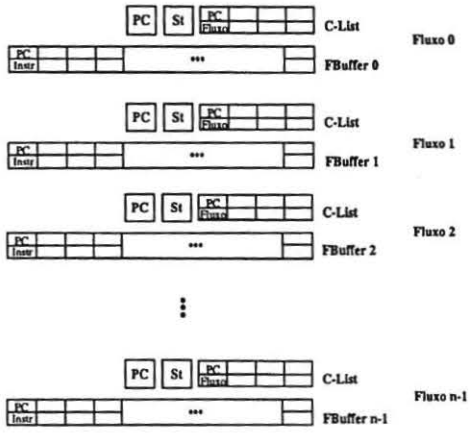

Figura 4: Estrutura de Buffers do novo Estágio de *Busca*

estrutura por algum "uxo e a *C-List* indica os "uxos filhos que se originam de um "uxo principal.

A *C-List* armazena o endereço de uma instrução de desvio e o identificador do seu "uxo filho, permitindo que o estágio de previsão redirecione a transferência automaticamente ao encontrar uma instrução de desvio.

5.2 Funcionamento da Arquitetura Multifluxo

Nesta subseção descreve-se o funcionamento dos estágios de *Busca* e *Previsão* modificados na arquitetura multi"uxo.

O estágio de *Busca* acessa instruções colocando-as no *Fetch Buffer*. Ao detectar uma instrução de desvio, durante a busca, um novo "uxo é criado e inicializado, se houverem recursos disponíveis.

A criação de um "uxo compreende a atualização da *C-List* do buffer corrente com o endereço (*PC*) do desvio detectado e com a identificação do buffer alocado para armazenar as instruções que serão buscadas no caminho alvejado por aquele desvio. Para cada desvio encontrado são possíveis dois caminhos. O caminho não tomado segue sendo buscado e armazenado no mesmo buffer do desvio em questão, poupando recursos. Já o caminho tomado passa a ser buscado no ciclo seguinte e armazenado no buffer alocado àquele "uxo.

A inicialização de um "uxo corresponde a inicialização do *PC* daquele buffer com o endereço alvo do desvio e do *status bit* indicando a ocupação da estrutura.

O estágio de previsão transfere as instrucões do "uxo principal para a fila de instruções como nas arquiteturas superescalares convencionais observando o tipo de cada instrução. Porém, ao encontrar uma instrução de desvio, a previsão é realizada e ao invés de redirecionar o estágio de *Busca*, em caso de previsão de desvio tomado, este estágio somente encadeia as instruções que estão no "uxo filho daquele desvio descartando as instruções adjacentes. O "uxo principal passa a ser então o "uxo filho do desvio em questão.

Caso a previsão seja de desvio não tomado, o ^uxo filho é descartado e as instruções que sucedem o desvio são transferidas para a fila de instruções. Quando um ^uxo é descartado, todos os ^uxos filhos originados a partir deste são também descartados de maneira recursiva liberando recursos para que novos ^uxos sejam especulados.

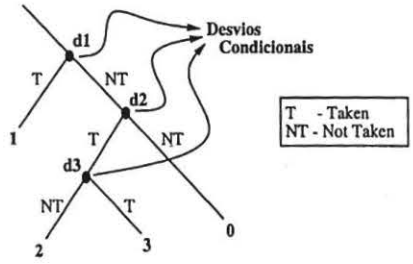

Figura 5: Ocorrencia de Desvios e Identificação de Fluxos

A figura 5 exemplifica a ocorrência de três desvios em um trecho de programa hipotético mostrando como os ^uxos são numerados e identificados. Para cada desvio que acontece, um novo ^uxo é criado e inicializado. As instruções que se encontram no caminho que seria previsto como não tomado fazem parte do mesmo ^uxo em que está o desvio, ou melhor, do ^uxo principal.

6 Desempenho do Modelo Multifluxo

Nesta seção analisa-se o desempenho do modelo multi^uxo baseado em dados extraídos através de diversas simulações. Os experimentos realizados simularam diferentes configurações para a máquina real e para a máquina multi^uxo. Os testes iniciaram variando-se a largura de busca de 2 até 8 instruções por ciclo para 2, 4 e 8 ^uxos no modelo multi^uxo. Para a máquina real simularam-se apenas a variação da largura de busca, já que somente um ^uxo pode ser acessado por vez. Os demais parâmetros foram mantidos iguais aos apresentados na seção 4.

O gráfico da figura 6 apresenta a porcentagem de ciclos em que ocorreu despacho nulo para as quatro máquinas diferentes. *Mulflux-2*, *Mulflux-4* e *Mulflux-8* correspondendo às simulações da máquina multi^uxo para 2, 4 e 8 ^uxos respectivamente, e *Real* às simulações da máquina real.

Observa-se que em todas as situações o modelo multi^uxo apresentou taxas menores de ocorrência de ciclos com despacho nulo. Na máquina *Real*, a taxa de despacho nulo diminui a medida em que a largura de busca aumenta, indo de 40,50% para largura de busca igual a dois para 39,30% em configurações com largura de busca igual a oito instruções por ciclo. As taxas de despacho nulo no modelo multi^uxo, apesar de serem menores do que na máquina *Real*, crescem à medida em que a largura de busca é aumentada. Isto ocorre devido ao aumento das previsões erradas e ao aumento da indisponibilidade de recursos. Outros

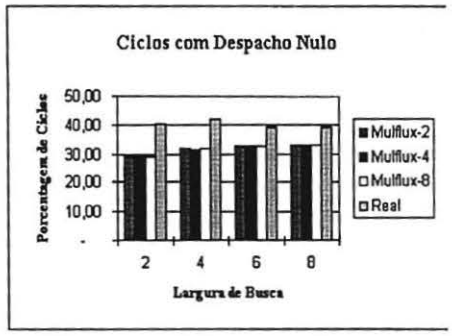

Figura 6: Comparação de Ciclos c/ Despacho Nulo

estudos estão sendo realizados objetivando a utilização de mecanismos mais eficientes de previsão de desvios e o balanceamento ideal da arquitetura. Com isto, pode-se diminuir então os ciclos com despacho nulo à medida em que a profundidade de especulação, na busca, é aumentada. Para os demais casos, utilizou-se apenas as configurações com dois fluxos devido à melhor relação custo e desempenho apresentada.

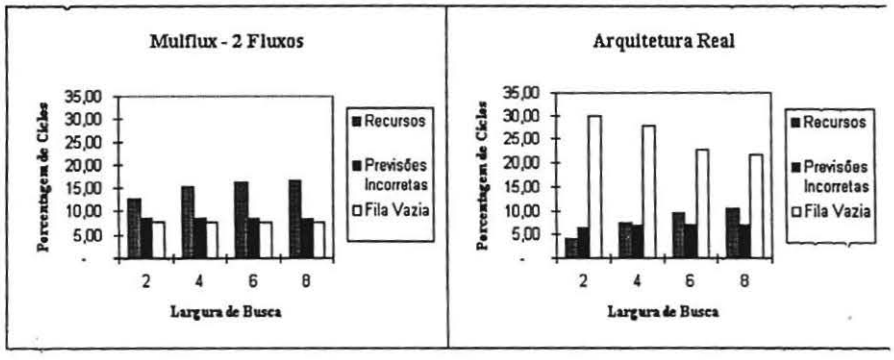

Figura 7: Componentes que Causam Ciclos com Despacho Nulo

Os gráficos da figura 7 apresentam a variação das componentes que causam despacho nulo nas duas máquinas. A soma das três componentes fornece a porcentagem de ciclos em que ocorreu despacho nulo.

A discrepância entre as componentes na máquina *Real* é notável. A ocorrência de fila vazia é a componente que mais contribui para a existência de despacho nulo, fator que não se verifica na máquina *Mulflux*. É de se esperar que a partir do momento em que existem mais instruções disponíveis para despacho, e posterior execução, o número de recursos disponíveis torne-se o fator determinante para a ocorrência de despacho nulo. Este fator é observado nitidamente na máquina *Mulflux* onde esta torna-se a maior componente.

A ocorrência de fila vazia na máquina *Real* decresce de 30,05%, com largura de busca igual a 2, para 21,79% com largura de busca igual a 8. Na máquina *Mulflux* esta taxa fica

entre 7,73% e 7,80% para as mesmas configurações.

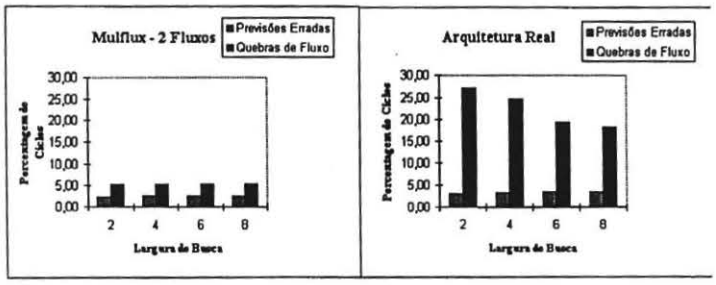

Figura 8: Fatores que Causam Esvaziamento da Fila de Instruções

Os gráficos da figura 8 provam a eficiência do modelo multi^uxo na redução da ocorrência de fila vazia devido as quebras de ^uxo. Na máquina *Mulflux* as quebras de ^uxo não ultrapassam os 5,24% enquanto que na máquina *Real* esta porcentagem chega aos 27,07%.

7 Conclusões

O *speed-up* de uma arquitetura superescalar é diretamente proporcional ao seu *IPC*, número médio de instruções executadas por ciclo. E, é esperado obter-se um *speed-up* proporcional ao número de unidades funcionais paralelas existentes no processador. No entanto, as constantes quebras de ^uxo ocasionadas pela previsão de desvios como tomados fazem com que a fila de instruções seja freqüentemente esvaziada e que não existam instruções para despacho. Logo, o *IPC* é reduzido drasticamente pelo esvaziamento da fila de instruções e o *speed-up* esperado não é alcançado em razão deste fator principal.

O modelo multi^uxo mostrou-se eficiente na redução da ocorrência de fila vazia. Encadeando instruções provenientes de ^uxos lógicos distintos, este modelo atenua o efeito causado pelas constantes quebras de ^uxo oriundas da previsão de desvios tomados. Este modelo possibilitou uma redução de aproximadamente 74,27% da ocorrência de fila vazia. O efeito causado pelas quebras de ^uxo sofreu redução em torno de 80,64% na máquina *Mulflux*. Outro aspecto interessante é que na máquina *Mulflux* os despachos nulos aumentaram à medida em que aumentou-se a largura de busca enquanto na máquina *Real* estes diminuiram. Isto explica-se devido ao aumento da indisponibilidade de recursos, devido ao maior ^uxo de instruções e ao aumento do número de previsões erradas.

Apesar de reduzir drasticamente a ocorrência de fila vazia, verificou-se que a diminuição dos despachos nulos não decresceu como esperado. Na máquina *Mulflux* as taxas de despacho nulo ficaram entre 28,99% e 33,11% enquanto na máquina *Real* estas taxas ficaram entre 39,30% e 40,50%. O aumento do ^uxo de instruções na fila de instruções causou um maior con^ito de recursos o que fez com que os ciclos com despacho nulo, no modelo mul^ux, não fossem reduzidos drasticamente. Configurações com um maior número de unidades funcionais podem oferecer melhores resultados.

A priori podemos detectar um outro gargalo no sistema ao empregar o modelo multi^uxo. A profundidade de especulação determina o número de desvios que podem ser especulativamente executados. Se o número médio de instruções aumenta devido a maior quantidade de instruções que tendem a preencher a fila de instruções, é muito provável que se tenha também um número maior de desvios a serem executados. A saturação da unidade de desvios pode retardar a execução das demais instruções e mascarar o potencial desempenho oferecido pelo modelo multi^uxo.

Outros estudos estão em andamento e serão apresentados oportunamente. A avaliação do impacto da implantação do modelo multi^uxo em arquiteturas reais, sob a ótica da cache de instruções, é um deles. O aumento do número de acessos à cache ou o acesso a pontos distantes em instantes próximos pode tornar o sistema de cache um gargalo. Outro estudo importante re^ete o balanceamento da arquitetura como um todo, a determinação do número ideal de unidades funcionais, largura de busca, largura de despacho, profundidade de especulação e tamanho de buffers estão sendo estudados e considerados como fatores fundamentais para o desenvolvimento das próximas etapas.

8 Agradecimentos

Este trabalho foi realizado no contexto do Projeto MULFLUX, patrocinado pelo CNPq através do programa ProTem-CC fase III (Processo Institucional No. 680096/95.7). Os autores, demais participantes e coolaboradores do projeto agradecem pelo suporte recebido para realização desta pesquisa.

Agradecimentos são destinados também ao Centro Nacional de Supercomputação (CESUP), localizado em Porto Alegre-RS, que financia parte deste projeto através do University Research & Development Grant Program oferecido pela Cray Research Inc.

Referências

[1] CHAVES FILHO, Eliseu Monteiro. *Arquiteturas Super Escalares: Efeito de Alguns Parâmetros sobre o Desempenho.* Rio de Janeiro: COPPE/UFRJ, 1994. (Tese de Doutorado)

[2] CHAVES FILHO, Eliseu M.; Fernandes, Edil S. T. On the Performance of Superscalar Processors. *Journal of the Brazilian Computer Society*, Rio de Janeiro, v.2, n.1, p. 38-48, July 1995.

[3] CHAVES FILHO, Eliseu M.; SOUZA, Alberto F.; SANTOS, Anna D.; SANTOS, Rafael R. *Uma Arquitetura Super Escalar com Múltiplos Fluxos de Instruções.* VIII SBAC-PAD - Simpósio Brasileiro de Arquitetura de Computadores. *anais...* Recife. PE : SBC, 1996.

[4] *PowerPC: Concepts, Architecture, and Design.* CHAKRAVARTY, Dipto; CANNON, Casey. McGraw Hill Inc., 1994.

[5] LEE, J. K. F.; SMITH, A. J. Branch Prediction Strategies and Branch Target Buffer Design. *Computer*, Los Alamitos, v.17, n.1, p.6-22, Jan. 1984.

[6] LILJA, David J. Reducing the Branch Penalty in Pipelined Processors. *Computer*, Los Alamitos, v.21, n.7, p.47-55, July 1988.

[7] LILJA, David J. Exploiting the Parallelism Available in Loops. *Computer*, Los Alamitos, v.27, n.2, p. 13-26, Feb. 1994.

[8] SANTOS, Rafael R.; *Arquiteturas Superescalares: Um Estudo sobre Dependências de Controle.* Porto Alegre: CPGCC da UFRGS, 1996. (Trabalho Individual).

[9] YEH, T.-Y.; PATT, Y. Two-Level Adaptive Training Branch Prediction. In: Annual International Symposium on Microarchitecture, 24., 1991. *Proceedings*. New York: ACM, 1991. p. 51-61.

[10] Pentium Pro Family Developer's Manual. Volume 1: *Specifications*. Intel Corporation, 1996.

[11] Pentium Pro Family Developer's Manual. Volume 1: *Programmer's Reference Manual*. Intel Corporation, 1996.

[12] SOHI, G. S.; BREACH, S. E.; VIJAYKUMAR, T. N. Multiscalar Processors. *Computer Architetcure News*, New York, v.23, n.2, p. 414-425, 1995.

[13] Sun Microsystems. *The SPARC Architecture Manual Version 7*. Mountain View, CA, 1987.

[14] TALCOTT, Adam R. *Reducing the Impact of the Branch Problem in Superpipelined and Superscalar Processors*. Santa Barbara: University of California, 1995 (Ph. D. Thesis).

[15] WALLACE, Steven D. *Performance Analisys of a Superscalar Architecture*, Irvine: University of California, 1993 (Ph. D. Thesis).

[16] WEAVER, David; GERMOND, Tom. *The SPARC Architecture Manual Version 9*. PTR Pretince Hall, Englewood Cliffs, New Jersey. 1994.

[17] *SPEC CPU 95 Technical Manual*, 1995. SPEC Steering Committe.