Avaliando Mecanismos de Tratamento de Dependências de Controle em Arquiteturas Super Escalares

Eliseu Monteiro Chaves Filho
Edil Severiano Tavares Fernandes
Programa de Engenharia de Sistemas e Computação
COPPE/Universidade Federal do Rio de Janeiro
Caixa Postal 68511
21945-970 Rio de Janeiro RJ
e-mail: {eliseu, edil}@cos.ufrj.br

Resumo

Dependências de controle, provocadas pelas instruções de desvio, podem representar uma séria limitação para o desempenho de arquiteturas super escalares. Neste trabalho, avaliamos o efeito de três diferentes mecanismos de tratamento de dependências de controle sobre o desempenho de uma arquitetura super escalar. No primeiro mecanismo, o despacho de instruções é interrompido em cada dependência de controle. No segundo mecanismo, uma unidade funcional especializada é incluída na arquitetura para tornar mais eficiente a execução das instruções de desvio. O terceiro mecanismo suporta a execução especulativa de instruções, permitindo a continuidade do despacho de instruções na presença de dependências de controle. Mostramos que ao bloquear o despacho estamos restringindo significativamente o desempenho da arquitetura, e que a introdução de uma unidade de desvios é uma maneira simples de melhorar o desempenho. Este trabalho também mostra os níveis de desempenho que podem ser obtidos com a execução especulativa de instruções.

Abstract

In this work, we investigate how the performance of superscalar architectures is affected by the mechanism used to handle control dependencies. Three different mechanisms are considered. The first mechanism handles control dependencies by stalling instruction dispatch. In the second mechanism, a specific functional unit is added to the architecture to speed the execution of branch instructions. In the third mechanism, instructions are executed speculatively, allowing instruction dispatch to continue even in the presence of control dependencies. We show that the solution of blocking instruction dispatch can seriously limit the performance, and that the use of a branch unit represents a simple alternative to increase performance. We also show the performance that can be achieved with speculative execution.

1 Introdução

Arquiteturas super escalares podem acessar, decodificar e executar múltiplas instruções simultaneamente. Com esta característica, as arquiteturas super escalares são capazes de manter a execução de mais de uma instrução por ciclo. Isto resulta em um ganho significativo de desempenho em relação à arquiteturas *pipeline* convencionais, que executam no máximo uma instrução por ciclo. Devido a este potencial, vários processadores de alto desempenho exibem arquiteturas super escalares. Alguns exemplos são: DEC Alpha 21064 [1,2], HP PA7100 [3], IBM RS/6000 [4], IBM-Motorola PowerPC [5,6,7,8], Intel Pentium [9] e Intel P6 [10], Motorola 88110 [11] e Sun SuperSPARC [12].

O real desempenho de uma arquitetura super escalar depende do grau de paralelismo a nível de instrução, ou seja, da existência de instruções no código do programa que possam ser executadas em paralelo. O paralelismo a nível de instrução é limitado por dois fatores: as dependências de dados e as dependências de controle [13]. Verifica-se que as dependências de controle impõem limitações mais fortes do que as dependências de dados. Por este motivo, em alguns processadores super escalares, grande parte do *hardware* é dedicada ao tratamento das dependências de controle [8,10]. Neste trabalho, avaliamos o efeito de três mecanismos de tratamento de dependências de controle sobre o desempenho de uma arquitetura super escalar. O primeiro representa a forma mais restritiva de tratamento, na qual o despacho de instruções é interrompido em cada dependência de controle. No segundo mecanismo, uma unidade funcional especializada é incluída na arquitetura para tornar mais eficiente a execução das instruções de desvio. O terceiro mecanismo suporta a execução especulativa de instruções, para permitir a continuidade do despacho de instruções mesmo na presença de dependências de controle. Estas três formas de tratamento das dependências de controle são encontradas em arquiteturas super escalares comerciais.

Este artigo está organizado em cinco seções. Na Seção 2 mostramos que as dependências de controle são mais limitantes para o desempenho do que as dependências de dados. Na Seção 3 descrevemos os modelos de arquitetura super escalares que incorporam os diferentes mecanismos mencionados acima. Na Seção 4 apresentamos e discutimos os resultados. Na Seção 5 finalizamos o artigo com alguns comentários adicionais.

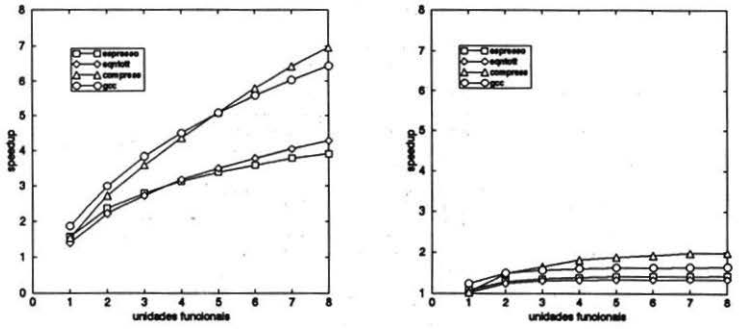
2 Dependências e seus Efeitos sobre o Desempenho

Uma dependência de dados se manisfeta quando duas instruções compartilham uma mesma variável. Existem três formas de dependências de dados. Na dependência verdadeira, uma instrução usa o valor de uma variável modificada por uma instrução precedente. Na anti-dependência, uma instrução modifica uma variável que será lida por uma instrução anterior. Na dependência de saída, uma instrução modifica uma variável que também será modificada por uma instrução anterior. Em todos estes casos, a execução de uma instrução fica subordinada ao término de uma instrução precedente, de forma que estas duas instruções devem ser executadas seqüencialmente, na ordem em que se encontram

no código. Em uma arquitetura super escalar, isto reduz o número médio de instruções executadas em paralelo a cada ciclo, prejudicando o desempenho.

As dependências de controle são provocadas pelas instruções de desvio. A instrução que deve ser executada após uma instrução de desvio é conhecida somente quando a instrução de desvio for completada. Cria-se assim uma relação de dependência, que obriga a execução seqüencial da instrução de desvio e da instrução seguinte. Novamente, a conseqüência é uma redução do número médio de instruções executadas por ciclo, e do desempenho.

Como uma arquitetura super escalar é capaz de despachar várias instruções simultaneamente, o número de instruções que deixam de ser despachadas e executadas por causa das dependências de controle torna-se muito grande. Por este motivo, as dependências de controle podem representar um obstáculo mais sério para o paralelismo de instrução e para o desempenho de uma arquitetura super escalar do que as dependências de dados. Os gráficos na Figura 1 suportam esta afirmação. Estes gráficos foram obtidos usando-se programas do conjunto SPECint92 [14], e mostram o desempenho de uma arquitetura super escalar em função do número de unidades funcionais. O desempenho é medido através do *speedup* em relação a uma arquitetura *pipeline* convencional. O gráfico na Figura 1(a) mostra o desempenho quando apenas as dependências de dados atuam. Este gráfico foi obtido com um tratamento perfeito das dependências de controle: quando uma instrução de desvio é acessada, o seu resultado é imediatamente determinado e o acesso e despacho das instruções seguintes prosseguem a partir do destino do desvio, sem interrupções. Desta forma, as dependências de controle se tornam transparentes. O gráfico na Figura 1(b) mostra o desempenho com as dependências de controle presentes. Quando uma instrução de desvio é encontrada, o despacho de novas instruções é interrompido até que a execução do desvio seja completada.



(a) (b)

Figura 1: EFEITOS DAS DEPENDÊNCIAS DE DADOS E DAS DEPENDÊNCIAS DE CONTROLE SOBRE O DESEMPENHO.

Com a presença apenas das dependências de dados, o desempenho continua a aumentar com o número de unidades funcionais, estabilizando apenas a partir de um número muito grande de unidades funcionais. O fato de serem necessárias várias unidades funcionais para saturar o desempenho indica que existe um grande volume de instruções que podem ser executadas em paralelo. Quando as dependências de controle são introduzidas, o desempenho estabiliza para um número bem pequeno de unidades funcionais, indicando a escassez de instruções para serem executadas em paralelo por unidades funcionais adicionais. Estes resultados mostram que o paralelismo de máquina embutido em uma arquitetura super escalar pode ser desperdiçado se as dependências de controle não forem devidamente tratadas.

3 Modelos de Arquiteturas Super Escalares

Para avaliar como o tratamento das dependências de controle afetam o desempenho, consideramos três diferentes arquiteturas super escalares. Estas arquiteturas possuem em comum um mesmo mecanismo para o tratamento das dependências de dados, baseado no algoritmo de Tomasulo [15]. O algoritmo de Tomasulo realiza um escalonamento dinâmico de instruções, resolvendo as dependências de dados sem interromper o despacho de instruções. Ao usar o algoritmo de Tomasulo, pretendemos minimizar os efeitos das dependências de dados, enfatizando apenas as limitações das dependências de controle e a atuação dos seus mecanismos de tratamento. Versões simplificadas do algoritmo de Tomasulo podem ser encontradas em alguns processadores super escalares, tais como o IBM-Motorola PowerPC 604 [8] e o Intel P6 [10]. A Figura 2 mostra a organização básica comum às arquiteturas super escalares avaliadas.

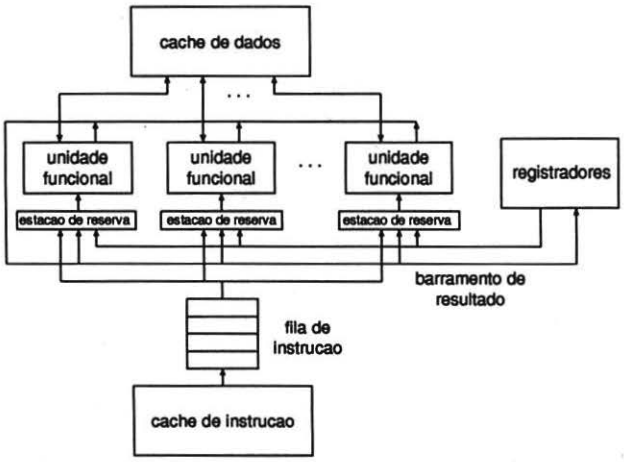


Figura 2: ORGANIZAÇÃO BÁSICA DAS ARQUITETURAS SUPER ESCALARES.

Cada unidade funcional possui uma ou mais estações de reserva, que armazenam o código e os operandos de instruções despachadas para a unidade funcional. A cada registrador está associado um bit de reserva e um campo contendo um identificador. No despacho de uma instrução, os bits de reserva dos registradores-fonte são testados. Se o bit estiver desativado, o registrador correspondente contém um dado válido, que é copiado para a estação de reserva alocada para a instrução sendo despachada. Se o bit estiver ativado, o registrador é o destino de uma instrução ainda não completada, e o identificador associado indica a estação de reserva na qual se encontra tal instrução. Este identificador é copiado para a estação de reserva da instrução sendo despachada.

Quando uma instrução é completada, o resultado produzido e o identificador de sua estação de reserva são transmitidos, através do barramento de resultados, para as estações de reserva e para os registradores. O identificador presente no barramento é comparado associativamente com os identificadores armazenados nas estações de reserva e com aqueles agregados aos registradores. O resultado é armazenado nas estações de reserva e no registrador que possuírem identificadores coincidentes. Identificadores coincidentes indicam que a instrução na estação de reserva usará o resultado como operando, ou que o registrador é o destino da instrução completada. Uma instrução torna-se apta para ser executada quando todos os seus operandos estiverem disponíveis na estação de reserva. Assim, após a transmissão de um resultado, uma ou mais instruções em estações de reserva podem entrar em execução. Instruções prontas são executadas de acordo com a sua ordem de chegada nas estações de reserva. A partir desta descrição, pode-se verificar que este mecanismo resolve automaticamente os três tipos de dependências de dados mencionados na Seção 2.

Instruções são despachadas em ordem a partir de uma fila de instruções. O algoritmo de despacho opera da seguinte forma: inicialmente, procura-se uma unidade funcional ociosa com uma estação de reserva livre. Se não for encontrada uma unidade funcional ociosa, ou se a unidade funcional ociosa estiver com todas as suas estações de reserva ocupadas, a instrução é despachada em *round-robin* em relação ao despacho anterior. Caso não seja possível encontrar uma estação de reserva livre, o despacho é suspenso até que os recursos necessários estejam disponíveis. A fila de instruções é preenchida a partir de uma memória *cache* de instruções. Para permitir o acesso de múltiplas instruções em um mesmo ciclo, a memória *cache* de instruções apresenta uma organização semelhante à encontrada no IBM RS/6000 [4]. As arquiteturas também incluem uma memória *cache* de dados.

A partir desta organização básica, são derivadas as diferentes arquiteturas super escalares, cujas particularidades são descritas a seguir.

3.1 A Arquitetura Bloqueante

A arquitetura bloqueante possui exatamente a mesma organização mostrada na Figura 2. Nesta arquitetura, o despacho das instruções subseqüentes a uma instrução de desvio é suspenso até que o desvio seja executado. Durante o intervalo de tempo em que o despacho permanece bloqueado, o acesso e a decodificação de instruções prossegue ao longo de um dos possíveis caminhos do desvio, previsto estaticamente. Quando a instrução de

desvio é completada, o despacho é retomado com as instruções acessadas antecipadamente, caso a previsão esteja correta. Se a previsão estiver incorreta, as instruções acessadas antecipadamente são descartadas, e o acesso e despacho são retomados com as instruções ao longo do caminho correto. Esta operação representa a forma mais simples, e também a mais restritiva, de tratamento das dependências de controle.

3.2 A Arquitetura Bloqueante com Unidade de Desvio

Esta arquitetura também possui a mesma organização apresentada na Figura 2. Mas, enquanto na arquitetura descrita em 3.1 as instruções de desvio são executadas por qualquer unidade funcional, agora os desvios são executados apenas por uma unidade funcional especializada, denominada unidade de desvio. Com a introdução desta unidade, a arquitetura passa a operar da seguinte forma. Instruções de desvio incondicional são executadas pela unidade de desvio no mesmo ciclo em que são despachadas. Instruções de desvio condicional também são executadas dentro do ciclo de despacho, se os bits de condição estiverem atualizados naquele ciclo. Caso contrário, o despacho das instruções subseqüentes é bloqueado até que o desvio seja executado. Desvios condicionais também são previstos estaticamente. Caso a previsão se revele incorreta, o despacho é bloqueado, as instruções acessadas antecipadamente são descartadas, e o despacho é retomado com as instruções acessadas no caminho correto.

Embora o despacho ainda possa ser bloqueado, isto acontece com uma freqüência menor do que na arquitetura anterior. Além disso, a unidade de desvio elimina (quando possível) o retardo entre o ciclo em que a instrução de desvio é despachada e o ciclo em que ela é completada, reduzindo o número total de ciclos durante os quais o despacho fica bloqueado. A inclusão de uma unidade de desvio é uma forma também simples, porém menos restritiva, de tratar as dependências de controle. Esta alternativa é usada no IBM RS/6000 (onde recebe o nome de *zero-cycle branches*) [4] e nos processadores IBM-Motorola PowerPC [5].

3.3 A Arquitetura com Execução Especulativa

Na arquitetura com execução especulativa, o despacho não é bloqueado quando uma instrução de desvio é encontrada. O despacho prossegue com as instruções ao longo de um dos ramos, previsto dinamicamente através de um *branch target buffer* [16]. Os resultados produzidos por estas instruções não alteram definitivamente o estado da arquitetura até que o resultado do desvio anterior seja conhecido. A arquitetura especulativa é derivada da organização básica apresentada na Figura 2 com o acréscimo de um mecanismo para manter o estado temporário criado pelas instruções executadas especulativamente. A Figura 3 mostra a organização da arquitetura com execução especulativa.

O conjunto de registradores futuros mantém o estado modificado pelas instruções executadas especulativamente, e fornece os operandos para as instruções que estão sendo despachadas. O conjunto de registradores reais guarda os estado definitivo da arquitetura. No despacho de uma instrução, uma entrada é inserida na fila de reordenação. Quando

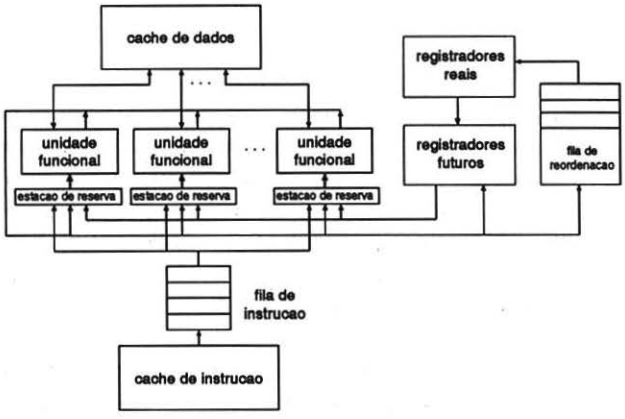


Figura 3: ORGANIZAÇÃO DA ARQUITETURA SUPER ESCALAR COM EXECUÇÃO ESPECULATIVA.

a instrução é completada, o seu resultado é transferido para as estações de reserva, para os registradores futuros e para a entrada correspondente na fila de reordenação. Os registradores reais são atualizados com o resultado somente quando a entrada correspondente à instrução atingir a cabeça da fila de reordenação. Assim, a fila de reordenação faz com que os registradores reais sejam atualizados na mesma ordem em que as instruções foram despachadas.

Quando uma instrução de desvio é acessada, o seu resultado é previsto pelo *branch target buffer* e o despacho continua ao longo do caminho previsto. Se o desvio foi previsto corretamente a execução prossegue normalmente, como descrito acima. Caso contrário, o despacho é bloqueado e todas as instruções nas estações de reserva, despachadas após o desvio, são descartadas. Estas instruções são indicadas por ponteiros nas entradas da fila de reordenação subseqüentes à entrada da instrução de desvio. Estas entradas são, depois, também descartadas. Quando a entrada da instrução de desvio alcançar a cabeça da fila, os registradores reais guardam um estado correto, modificado apenas pelas instruções despachadas antes do desvio. Neste momento, os registradores reais são copiados para os registradores futuros, cancelando as modificações efetuadas pelas instruções executadas indevidamente. Após a correção dos registradores futuros, o despacho é retomado com as instruções acessadas a partir do destino correto do desvio.

A execução especulativa de instruções é a maneira menos restritiva de tratamento das dependências de controle, pois o bloqueio do despacho de instruções é bastante reduzido. A fila de reordenação com registradores futuros foi originalmente proposta para garantir interrupções precisas em arquiteturas com execução fora-de-ordem [17], e foi aqui adaptada para suportar execução especulativa. Processadores como o IBM-Motorola PowerPC 604 [8] e o Intel P6 [10] também usam soluções baseadas em uma fila de reordenação.

4 Resultados

4.1 Ambiente Experimental

Em nossos experimentos, o desempenho das arquiteturas super escalares foi avaliado medindo-se o *speedup* em relação a uma arquitetura *pipeline* escalar. A arquitetura Sun SPARC [18] foi escolhida como referência, pois esta arquitetura obteve sucesso em várias aplicações de alto desempenho. Foi desenvolvido um simulador que segue a implementação da arquitetura SPARC encontrada no processador Fujitsu MB89600 [19]. Este simulador contabiliza o número de ciclos gastos na execução de um certo programa, e produz dois arquivos (*traces*) que registram a história de execução das instruções de desvio e das instruções de acesso a memória.

Foram também construídos simuladores para cada uma das arquiteturas super escalares descritas. Estes simuladores são do tipo *trace-driven*, e recebem como entrada os arquivos de história gerados pelo simulador SPARC. O *speedup* é calculado como a razão entre o número de ciclos consumidos pela arquitetura SPARC e o número de ciclos consumidos pela arquitetura super escalar. Além do *speedup*, os simuladores das arquiteturas super escalares fornecem várias outras estatísticas, como por exemplo, o número médio de instruções completadas por ciclo e a utilização das unidades funcionais.

Como programas de teste, foram usados alguns programas do conjunto SPECint92: *espresso* (minimizador de expressões booleanas), *eqntott* (gerador de mapas de PLAs), *compress* (utilitário de compressão de arquivos) e *gcc* (compilador GNU C). Estes programas foram compilados com o compilador UNIX C para a arquitetura SPARC sob o sistema operacional SunOS 4.1.1. Foram usados arquivos de história que cobrem a execução de 10 milhões de instruções destes programas.

4.2 Configurações das Arquiteturas

Nos experimentos realizados, as arquiteturas descritas na seção anterior foram configuradas da seguinte forma. A cada unidade funcional estão associadas três estações de reserva. Resultados adicionais aos aqui apresentados [20] indicam que não mais do que três estações de reserva são usadas por unidade funcional. A fila de instruções possui 24 posições. A memória *cache* de instruções possui um tamanho de 32 Kbytes, sendo do tipo associativa por conjunto com linhas de 64 bytes e quatro linhas por conjunto. A memória *cache* de instruções permite o acesso de até oito instruções por ciclo. A memória *cache* de dados possui o mesmo tamanho e organização da memória *cache* de instruções. Na arquitetura com execução especulativa, o *branch target buffer* possui 256 entradas, e a fila de reordenação possui 32 entradas.

4.3 Avaliação

Os gráficos na Figura 4 mostram o *speedup* em função do número de unidades funcionais. As curvas em cada gráfico correspondem a diferentes larguras de despacho. Definimos a largura de despacho como sendo o número máximo de instruções que podem ser

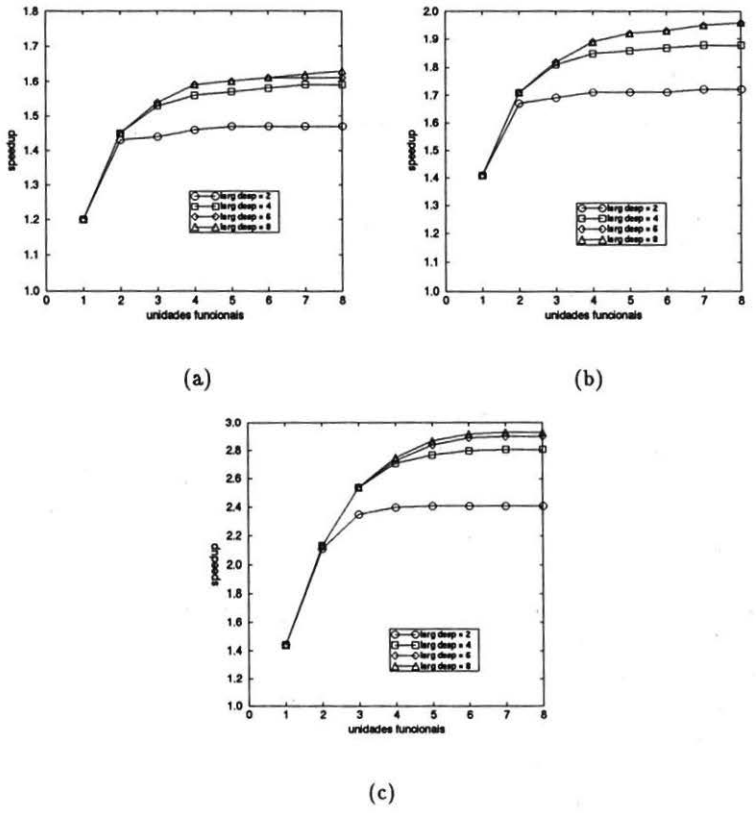


Figura 4: DESEMPENHO DAS ARQUITETURAS COM DIFERENTES MECANISMOS DE TRATAMENTO DAS DEPENDÊNCIAS DE CONTROLE.

despachadas simultaneamente. Os gráficos nas partes (a), (b) e (c) da figura mostram o desempenho das arquiteturas bloqueante, bloqueante com unidade de desvio e especulativa, respectivamente. A Tabela I fornece alguns dados adicionais, que ajudam a explicar o comportamento observado nos gráficos da Figura 4. A porcentagem de ciclos sem despacho e o número médio de instruções completadas por ciclo (ipc) foram medidos usando-se configurações com largura de despacho de oito instruções e com oito unidades funcionais. A utilização média das unidades funcionais foi obtida usando-se configurações com largura de despacho de oito instruções e com quatro unidades funcionais.

Tabela I: Algumas taxas de utilização das arquiteturas avaliadas.

Arquitetura	Ciclos sem Despacho	Utilização Média Unidades Funcionais	Instruções por Ciclo (ipc)
Bloqueante	80%	22%	0.9
c/ Unidade de Desvio	75%	26%	1.2
Especulativa	30%	42%	2.1

Como esperado, a arquitetura bloqueante apresenta um baixo desempenho. O gráfico na Figura 4(a) mostra que, mesmo para uma largura de despacho de oito instruções, o desempenho estabiliza-se com apenas três unidades funcionais. Como mencionado na Seção 2, tal comportamento sugere que um volume muito pequeno de instruções fica disponível para as unidades funcionais. De fato, para uma largura de despacho de oito instruções, em 80% dos ciclos nenhuma instrução é despachada (vide Tabela I), fazendo com que cada unidade funcional fique ocupada em apenas 22% dos ciclos. Como resultado final, em média apenas 0.9 instruções são completadas por ciclo.

Essa baixa utilização dos recursos da arquitetura é resultado de dois fatores: a freqüência de bloqueios do despacho e o intervalo de tempo durante o qual o despacho permanece bloqueado. Na segunda arquitetura, a unidade de desvio diminui estes dois fatores. Agora, o despacho é bloqueado somente nas instruções de desvio condicionais que não encontram os bits de condição atualizados. O gráfico na Figura 4(b) e os dados na Tabela I mostram que o desempenho é bastante sensível a estes fatores. Com a unidade de desvio obtém-se um aumento de 20% no desempenho, embora a proporção de ciclos nos quais nenhuma instrução é despachada tenha sido reduzida de apenas 5%. A capacidade de despachar múltiplas instruções faz com que um modesto aumento nos ciclos com despacho resulte em um ganho comparativamente maior no desempenho final da arquitetura. É justamente por este motivo que a introdução da unidade de desvio se revela tão eficaz. Devido à menor freqüência dos bloqueios do despacho, o volume de instruções disponíveis torna-se maior, tornando compensadora a inclusão de um número maior de unidades funcionais na arquitetura. Para configurações com quatro unidades funcionais, a utilização média de cada unidade funcional é de 26%, resultando em um ipc médio de 1.2 instruções por ciclo.

A Figura 4(c) mostra que a execução especulativa de instruções aumenta mais ainda o desempenho de uma arquitetura super escalar. Isto acontece porque agora o despacho é interrompido quando o resultado de um desvio é previsto incorretamente (na realidade, este é um dos fatores que provoca o bloqueio do despacho, como discutido logo a seguir). Em nossos experimentos, verificou-se uma taxa média de acerto de previsão de 85%. Usando um *branch target buffer*, pode-se esperar taxas de acerto na faixa de 70% a 90% [18]. A baixa freqüência dos bloqueios reduz para 30% a proporção de ciclos onde nenhuma instrução é despachada. Com um maior volume de instruções disponíveis, a utilização das unidades funcionais chega a 42%. O resultado final da melhor utilização dos recursos é a taxa ipc 2.1 instruções completadas por ciclo, bem superior às taxas obtidas com as duas

versões da arquitetura bloqueante.

Na arquitetura especulativa, o despacho é bloqueado em duas situações. A primeira delas, já mencionada anteriormente, acontece quando um desvio é previsto incorretamente. A segunda condição ocorre quando a profundidade de especulação é ultrapassada. Definimos a profundidade de especulação como sendo o número máximo de instruções de desvio através das quais a execução especulativa pode prosseguir. Por exemplo, uma profundidade de dois desvios significa que no máximo duas instruções de desvio podem estar pendentes, aguardando execução. Se uma terceira instrução de desvio for encontrada antes destas duas instruções serem executadas, o despacho será bloqueado. A profundidade de especulação é determinada por fatores relacionados com a configuração da arquitetura, tais como o tamanho da fila de reordenação. As arquiteturas PowerPC 603 [7] e PowerPC 604 [8] possuem profundidades de especulação de uma e duas instruções de desvio, respectivamente.

A profundidade de especulação é tão importante para o desempenho quanto a taxa de acerto do *branch target buffer*, pois ela também determina a freqüência de bloqueio do despacho. Note que uma pequena profundidade de especulação leva a um maior número de bloqueios, principalmente em configurações com largura de despacho ampla. Por outro lado, uma grande profundidade de especulação possibilita que um maior número de instruções sejam executadas indevidamente, aumentando o custo de descartar estas instruções. O gráfico na Figura 4(c) foi obtido com uma profundidade de especulação de quatro desvios. O gráfico na Figura 5 mostra o desempenho da arquitetura especulativa para outras profundidades de especulação. Este gráfico foi obtido para configurações de arquitetura com largura de despacho de oito instruções.

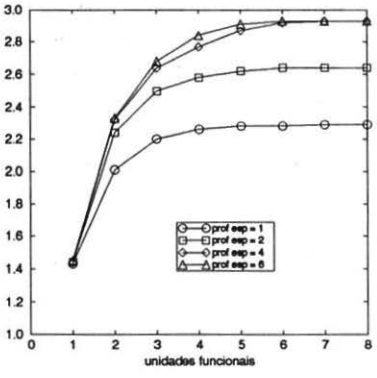


Figura 5: DESEMPENHO DA ARQUITETURA ESPECULATIVA EM RELAÇÃO À PROFUNDIDADE DE ESPECULAÇÃO.

Como esperado, o desempenho diminui para profundidades de especulação pequenas, pelo motivo acima exposto. No entanto, a partir de uma profundidade de quatro desvios,

não há um ganho significativo de desempenho quando aumentamos a profundidade de especulação. A partir deste ponto, a profundidade de especulação deixa de ser o fator predominante na freqüência de bloqueios do despacho, e a taxa de acertos na previsão de desvios passa a ser mais importante.

5 Conclusão

Este trabalho mostra que as dependências de controle possuem um papel determinante no desempenho das arquiteturas super escalares. Os resultados aqui apresentados servem como subsídio para uma avaliação do potencial desempenho de uma arquitetura super escalar, tendo em vista o mecanismo de tratamento das dependências de controle. Os resultados indicam que um processador com execução especulativa de instruções pode alcançar um desempenho quase duas vezes superior ao de um processador que simplesmente bloqueia o despacho a cada desvio. Neste último, o bloqueio freqüente do despacho limita severamente o volume de instruções disponíveis para execução. Como resultado verifica-se um baixo aproveitamento dos recursos, e o fator ipc pode ficar aquém da fronteira de uma instrução por ciclo. Ao contrário, em uma arquitetura com execução especulativa, a freqüência de bloqueios é reduzida ao mínimo, possibilitando uma exploração mais eficaz do paralelismo de instrução disponível no código do programa. A boa utilização dos recursos e a taxa ipc alcançados com a execução especulativa justificam o *hardware* empregado para implementar tal mecanismo. A inclusão da unidade de desvio é uma alternativa intermediária entre os dois extremos acima. A capacidade de executar desvios no mesmo ciclo do despacho é bastante eficaz em reduzir a proporção de ciclos sem despacho.

O objetivo central neste trabalho foi avaliar diferentes formas de como tratar as dependências de controle. Uma questão adjacente está em como este tratamento afeta o balanceamento da arquitetura super escalar. Dependendo de como as dependências de controle sejam resolvidas, é vantajoso ou não dotar a arquitetura de um maior número de recursos. Por exemplo, em arquiteturas que adotam o simples bloqueio do despacho, não há ganho algum em incluir mais do que três unidades funcionais. Como visto, a forma como as dependências de controle são tratadas determinam, em grande parte, a utilização dos recursos. Assim, a configuração de uma arquitetura super escalar deve ser considerada dentro de um contexto que depende do mecanismo escolhido para resolver as dependências de controle. Esta questão é explorada em maiores detalhes em trabalhos relacionados ao aqui apresentado [20,21].

6 Referências

[1] Sites, R. L., Sites, R. L., *Alpha AXP Architecture*, Communications of the ACM (36)2, February 1993, pp. 33-44.

[2] McLellan, E., *The Alpha AXP Architecture and 21064 Processor*, IEEE Micro (13)3, June 1993, pp. 36-47.

[3] Asprey, T. *et al.*, *Performance Features of the PA7100 Microprocessor*, IEEE Micro (13)3, June 1993, pp. 11-21.

[4] Grohoski, G. F., *Machine Organization of the IBM RISC System/6000 Processor*, IBM Journal of Research and Development (34)1, January 1990, pp. 37-58.

[5] Diefendorff, K., *History of the PowerPC Architecture*, Communications of the ACM (37)6, June 1994, pp. 28-33.

[6] Becker, M. *et al.*, *The PowerPC 601 Microprocessor*, IEEE Micro (13)5, October 1993, pp. 54-68.

[7]Burgess, B. *et al.*, *The PowerPC 603 Microprocessor*, Communications of the ACM (37)6, June 1994, pp. 34-42.

[8] Song. P. S.,*et al.*, *The PowerPC 604 Microprocessor*, IEEE Micro (14)5, October 1994, pp. 8-17.

[9] [Alpert 93] Alpert, D., D. Avnon, *Architecture of the Pentium Microprocessor*, IEEE Micro (13)3, June 1995, pp.11-21.

[10] Colwell. R. P., Steck, R. E., *A 0.6um BiCMOS Processor Employing Dynamic Execution*, Proceedings of the International Solid State Circuits Conference, February 1995.

[11] Diefendorff, K., M. Allen, *Organization of the Motorola M88110 Superscalar RISC Microprocessor*, IEEE Micro (12)2, April 1992, pp. 40-63.

[12] Blanck, G., S. Krueger, *The SuperSPARC Microprocessor*, Proceedings of the COMPCON, 1992, pp.136-141.

[13] Patterson, D., Hennessy, J., *Computer Architecture: A Quantitative Approach*, Morgan Kauffman, Palo Alto, CA, 1990.

[14] SPEC Steering Committee, *SPEC INT92 Release V1.1 Technical Manual*, 1992.

[15] Tomasulo, R. M., *An Efficient Algorithm for Exploiting Multiple Arithmetic Units*, IBM Journal of Research and Development (11)1, January 1967, pp. 25-33.

[16] Lee, J. K. F., A. J. Smith, *Branch Prediction Strategies and Branch Target Buffer Design*, IEEE Computer (17)1, January 1984, pp. 6-22.

[17] Smith, J. E., A. R. Pleszkun, *Implementing Precise Interrupts in Pipelined Processors*, IEEE Transactions on Computers (37)5, May 1988, pp. 562-573.

[18] Garner, R. B. *et al.*, *The Scalable Processor Architecture (SPARC)*, Proceedings of the COMPCON, 1988, pp. 278-283.

[19] Namjoo, M. *et al.*, *CMOS Gate Array Implementation of the SPARC Architecture*, Proceedings of the COMPCON, 1988, pp. 10-13.

[20] Chaves Filho, Eliseu M., *Arquiteturas Super Escalares: Efeito de Alguns Parâmetros sobre o Desempenho*, Tese de Doutorado, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, 1994.

[21] Chaves Filho, Eliseu M., E.S.T. Fernandes, *On the Performance of Superscalar Processors*, a ser publicado no Journal of the Brazilian Computer Society.