# Avaliação de Modelos de Acesso à Memória de Dados em Arquiteturas Super Escalares

Emmanuel M. Pereira, Eliseu M. Chaves Filho

[1] Programa de Engenharia de Sistemas e Computação
COPPE/Universidade Federal do Rio de Janeiro
Caixa Postal 68511 21945-970 Rio de Janeiro, RJ, Brasil
{emmanuel, eliseu}@cos.ufrj.br

*Abstract*—

**The execution of data memory access instructions is an important issue for processor performance, due to the high frequency of such instructions. This work focus on the execution of memory access instructions in superscalar architectures. Several different memory access models are evaluated, ranging from a simple mechanism in which memory access instructions are executed sequentially in order, to a mechanism which executes memory access instructions in pipeline, out of order and speculatively.**

*Keywords*— **ILP, Superscalar Architectures**

## I. Introdução

A execução eficiente de instruções de acesso à memória de dados (instruções LOAD e STORE) vem assumindo importância na exploração do paralelismo no nível de instrução. Estas instruções contribuem com uma alta porcentagem do total das instruções executadas por um programa [1]. Por este motivo, é essencial aproveitar o paralelismo dentro desta classe de instruções.

A exploração de paralelismo entre instruções de memória requer mecanismos adicionais aos utilizados pelos demais tipos de instruções que acessam apenas registradores internos do processador. Este trabalho avalia o desempenho obtido com cinco diferentes modelos de acesso à memória de dados, alguns destes encontrados em microprocessadores comerciais. Tais modelos apresentam complexidade de implementação crescente. Duas perguntas se colocam: (1) qual o ganho obtido com um modelo que incorpora um certo conjunto de facilidades no acesso à memória de dados? (2) o ganho obtido com tal modelo justificaria o seu custo de implementação? O objetivo deste trabalho é fornecer subsídios para a solução destas duas questões.

O restante deste artigo está organizado da seguinte forma. A Seção II descreve a arquitetura base usada como moldura para os diversos modelos de acesso à memória aqui avaliados. A Seção III descreve os modelos de acesso à memória considerados. A Seção IV discute a metodologia empregada, enquanto na Seção V são apresentados e discutidos os resultados experimentais. A Seção VI conclui este artigo.

## II. A Arquitetura Base

A arquitetura base aqui considerada guarda similaridades com a arquitetura do processador PowerPC 620 [2]. A arquitetura base, juntamente com um determinado mecanismo de acesso à memória, formam uma arquitetura derivada específica. A arquitetura base está organizada em um *pipeline* com sete estágios.

O estágio de busca de instruções (BI) acessa instruções em uma memória *cache* do tipo associativa por conjunto com 32 KBytes. O estágio de previsão de desvios (PD) prevê o resultado de instruções de desvio usando uma BHT (*Branch History Table*) com contador saturável de 2 *bits* [1].

O estágio de decodificação de instruções (DC) coloca instruções em uma *fila de despacho*. O estágio de despacho de instruções (DP) retira instruções desta fila e as envia para estações de reserva. Este estágio executa parte do algoritmo de Tomasulo [1]. Para cada instrução despachada, o estágio PD aloca uma entrada em uma *fila de reordenação* [1].

O estágio de escalonamento de instruções (ES) executa a segunda fase do algoritmo de Tomasulo, verificando quais instruções nas estações de reserva podem ser enviadas para as unidades funcionais. O estágio de execução de instruções (EX) inclui múltiplas unidades de inteiros e uma unidade de memória. Ao considerarmos diferentes modelos de acesso à memória, estamos na realidade modificando a operação da unidade de memória. A unidade de memória acessa dados em uma memória *cache* do tipo associativa por conjunto, com 32 Kbytes.

O estágio de armazenamento de resultados (RS) realiza a fase final do algoritmo de Tomasulo, enviando resultados e seus *tags* para as estações de reserva e para o conjunto de registradores futuros [1]. O estágio RS também retira instruções da fila de reordenação, escrevendo no conjunto de registradores reais os resultados armazenados nesta fila. Ao encontrar uma instrução de desvio, o estágio RS verifica se resultado do desvio foi previsto correta ou incorretamente. Neste último caso, o estágio RS invalida todas as instruções que se encontram nas estações de reserva e nas entradas da fila de reordenação, corrige o conteúdo dos registradores fu-

turos a partir dos registradores reais e redireciona o estágio de busca para o destino correto.

## III. Modelos de Acesso à Memória

Neste trabalho foram considerados cinco diferentes modelos de acesso à memória, cada um deles adicionando uma facilidade que possibilita uma melhor exploração do paralelismo entre as instruções de acesso à memória. Estes modelos (numerados de 0 a 4) foram obtidos a partir de combinações das seguintes alternativas:

- Tipo da unidade funcional de memória: unidade seqüencial ou *pipeline*;
- Escalonamento das instruções de acesso à memória: escalonamento em ordem ou fora de ordem;
- Ordenação dos acessos à memória: acessos em ordem ou fora de ordem;
- Especulação nos acessos à memória: acessos não especulativos ou especulativos (acessos realizados à sombra de um desvio não resolvido);

Cada modelo é embutido na arquitetura super escalar base descrita na seção anterior. Assim, cada um deles possui um certo número de estações de reserva, que recebem as instruções de memória despachadas. O despacho é sempre em ordem, independente do tipo de instrução (característica da arquitetura base). Instruções de inteiros podem ser escalonadas fora de ordem (outra característica da arquitetura base). No caso das instruções de acesso à memória, a política de escalonamento depende do particular modelo.

### A. Modelo 0

O Modelo 0 é baseado em uma unidade funcional seqüencial. O escalonamento de instruções de memória acontece em ordem. Os acessos à memória *cache* de dados são em ordem e não especulativos. A latência de acesso à memória considerada foi de três ciclos, considerando-se um ciclo para o cálculo do endereço efetivo, outro para a verificação de *hit* ou *miss* e o terceiro para o acesso à *cache*. Por ser o mais restritivo, o Modelo 0 fornece o limite inferior de desempenho.

### B. Modelo 1

No Modelo 1, a unidade de memória seqüencial é substituída por uma unidade *pipeline*, introduzindo paralelismo temporal. O escalonamento de instruções de memória continua em ordem. Os acessos à memória *cache* de dados ainda são em ordem e não especulativos. O *pipeline* da unidade de memória possui três estágios:

- Estágio EA: calcula o endereço efetivo da posição de memória a ser acessada;
- Estágio HM: recebe o endereço efetivo e verifica se o acesso resulta em *hit* ou *miss*. Se o acesso resulta em *hit*, a instrução prossegue para o estágio seguinte. No caso de *miss*, o *pipeline* é paralisado e a instrução permanece no estágio HM até que a linha seja carregada na *cache*. Em conseqüência, o escalonamento das instruções de memória é temporariamente suspenso;
- Estágio AC: realiza o acesso propriamente dito à memória *cache*.

O *pipeline* de três estágios apresenta uma latência de três ciclos, a mesma do modelo seqüencial, permitindo assim uma comparação justa com o Modelo 0.

### C. Modelo 2

O Modelo 2 e os modelos subseqüentes são baseados na mesma estrutura de *pipeline* com três estágios. No Modelo 2, o escalonamento de instruções de memória permanece em ordem. Os acessos à memória *cache* de dados passam a ser fora de ordem, porém ainda continuam sendo não especulativos. Para suportar acessos fora de ordem, é utilizada uma memória *cache* não bloqueante do tipo *hit under multiple misses* [1]. Além disso, é introduzida uma fila de acessos à memória, denominada MAQ (*Memory Access Queue*). Ao encontrar um acesso com *miss*, o estágio HM coloca a instrução correspondente na MAQ. No entanto, acessos que resultam em *hit* são enviados diretamente para o estágio AC, ultrapassando assim as instruções na MAQ. As instruções com *miss* permanecem enfileiradas na MAQ enquanto os respectivos blocos são carregados na *cache*. Finalmente, também é incorporado um mecanismo de repasse (*forwarding*) para resolução de dependências verdadeiras entre as instruções de memória.

Com acessos fora de ordem, seria necessário um mecanismo para remover as ambigüidades [3], e assegurar que dependências de dados entre instruções de memória fossem respeitadas. No entanto, no Modelo 2, as dependências de dados são naturalmente satisfeitas. Note que alterações de ordem ocorrem sempre entre uma instrução que resultou em *hit* e instruções que resultaram em *miss* e por isso se encontram na MAQ. Nenhuma instrução $i$ que esteja na MAQ acessa a mesma posição referenciada por uma instrução $j$ com *hit*, que passou à frente. Se este fosse o caso, a instrução $i$ não seria um *miss* e não estaria na MAQ, tendo acessado a *cache* antes de $j$ (ou seja, em ordem). No caso de duas instruções prontas, uma na MAQ e outra no estágio HM, a instrução na MAQ tem prioridade de acesso por ser anterior.

### D. Modelo 3

Neste modelo o escalonamento de instruções de memória continua em ordem. Como no Modelo 2, os acessos à memória *cache* de dados podem acontecer fora de ordem. No entanto, agora são permitidos acessos LOAD especulativos. O estágio HM testa se uma instrução STORE é especulativa. Se for, esta instrução é inserida na MAQ, sendo retirada apenas

quando deixa de ser especulativa, uma vez que instruções STORE especulativas não podem acessar a memória, sob pena de alterar indevidamente o estado da computação. Instruções STORE presentes na MAQ também repassam dados para instruções LOAD com o mesmo endereço de acesso.

Todas as instruções presentes na MAQ são tratadas somente quando chegam na frente da fila. Desta forma preserva-se a ordem na qual as instruções foram despachadas.

### *E. Modelo 4*

Neste modelo, o escalonamento de instruções passa a ser fora de ordem. Neste caso, é realmente necessário o uso de um mecanismo para remover as ambigüidades. Para tanto, foi introduzida uma fila denominada MRQ (*memory reorder queue*), que registra a ordem de despacho das instruções de memória. No Modelo 4 foi abandonada a MAQ, uma vez que a sua natureza de fila é adequada apenas para os mecanismos com escalonamento em ordem. Em seu lugar, foram introduzidas três estruturas, controladas pelo estágio HM:

- MAF (*miss address file*): armazena instruções que resultaram em *miss*; e instruções STORE com *hit* que, por serem especulativos, não podem prosseguir imediatamente para o estágio AC do *pipeline*;
- STF (*store file*): recebe instruções STORE com dependências de saída em relação a algum STORE anterior.
- LDF (*load file*): recebe instruções LOAD com dependências verdadeiras em relação a algum STORE anterior;

A remoção de ambigüidade nos acessos à memória ocorre da seguinte forma. Ao receber uma instrução, o estágio HM faz uma busca associativa na MRQ, procurando algum STORE anterior que referencie a mesma posição de memória. Caso não exista uma dependência, o estágio HM passa a instrução para o estágio AC. Caso contrário, se é detectada uma dependência em relação a um STORE anterior, o estágio HM armazena a instrução no LDF ou STF caso a instrução seja, respectivamente, um LOAD ou um STORE. Também é considerado que existe uma dependência se a MRQ contém alguma instrução STORE com referência de memória não resolvida.

A operação acima refere-se à parte de *detecção* de dependência. Para *resolver* a dependência, o mecanismo opera como explicado a seguir. Quando uma instrução de memória $i$ chega à frente da MRQ, todas as instruções de memória anteriores já acessaram a *cache*. Assim, alguma dependência existente entre a instrução $i$ e alguma instrução anterior já foi satisfeita. Por este motivo, ao chegar à frente da MRQ, uma instrução inserida no LDF ou STF pode retornar ao *pipeline*. A implementação deste esquema consiste em, a cada ciclo, fazer uma busca associativa da instrução na frente da MRQ dentre as que estão no LDF ou STF. A instrução selecionada retorna para o estágio AC do *pipeline*.

## IV. Metodologia

Foi desenvolvido um programa simulador *trace-driven* específico para cada combinação da arquitetura base com um certo mecanismo de acesso à memória. Para obter os *traces*, foi empregado um simulador *execution-driven* que reproduz a operação de uma implementação *pipeline* da arquitetura SPARC [4].

Além do arquivo de *trace*, os simuladores super escalares também recebem como entrada o arquivo objeto do programa de teste e um arquivo de parâmetros contendo as informações de configuração da arquitetura. Neste trabalho, estaremos variando a largura de despacho (i.e., o número máximo de instruções que podem ser despachadas em um mesmo ciclo) e o número de estações de reserva de memória. Serão consideradas larguras de despacho de 2, 4, 6 e 8 instruções/ciclo e 2, 4, 6 e 8 estações de reserva na unidade de memória. Todas as configurações consideradas possuem quatro unidades de inteiros, cada uma com 8 estações de reserva.

Três métricas principais serão utilizadas neste trabalho:

- *Número médio de instruções por ciclo (IPC).* O IPC fornece o desempenho final da arquitetura.
- *Número médio de instruções de memória por ciclo (IPCmem).* O IPCmem mede a vazão do mecanismo de acesso à memória. Ele é dado pela razão entre o número de instruções de memória executadas e o número total de ciclos. É importante notar que o IPCmem também é dependente da densidade de instruções de memória exibido pelo programa.
- *Taxa de bloqueio do despacho.* Esta medida fornece a proporção do total de ciclos de execução no qual nenhuma instrução foi despachada para as estações de reserva. Estaremos interessados nos bloqueios do despacho provocados por ineficiências do mecanismo de acesso à memória.

Neste trabalho foram usados os seguintes programas do conjunto SPECint [5]: `espresso`, `eqntott`, `go`, `compress`, `gcc`, `m88ksim`, `ijpeg`, `vortex`. Estes programas foram compilados com o compilador GNU gcc-2.7.2, com nível de otimização máximo, para o sistema operacional SunOS 4.1.4. Para cada um destes programas de teste, foram utilizados nos experimentos *traces* que cobrem a execução de 20 milhões de instruções.

## V. Resultados e Análise

Como mencionado, o Modelo 0 estabelece os limites inferiores de desempenhos. É importante também que se verifique o limite superior de desempenho, que seria alcançado por um modelo ideal. Este Modelo Ideal, seria capaz de (1) realizar uma perfeita remoção de ambigüidade, antecipando a existência de dependências entre as instruções de memória

já no momento do despacho destas instruções e (2) realizar acessos em um único ciclo, nos casos de *hit* na memória *cache* de dados.

Mecanismos reais tratam a remoção de ambigüidade de maneira conservadora. No momento do despacho de uma instrução de memória, caso exista alguma instrução com referência não resolvida, a execução da instrução de memória permanece bloqueada enquanto o endereço efetivo não for calculado. Isto acontece mesmo que não exista realmente uma dependência através de memória. Ao contrário, o Modelo Ideal usaria um *oráculo* para, no despacho da instrução de memória, obter o endereço efetivo da posição a ser acessada. A execução da instrução de memória seria bloqueada apenas se de fato existisse uma dependência através de memória em relação a uma instrução anterior. Note que, mesmo no Modelo Ideal, é necessário que o bloqueio aconteça de forma que as dependências sejam respeitadas.

O Modelo Ideal também foi implementado na arquitetura base. A Tabela I resume os valores de IPC, IPCmem e da taxa de bloqueio obtidos com este modelo. Note que o valor do IPCmem é menor do que o limite superior teórico (1.0 instrução de memória/ciclo), não devido a alguma ineficiência do modelo, mas sim porque a densidade de instruções de memória não é suficiente para que este limite seja alcançado.

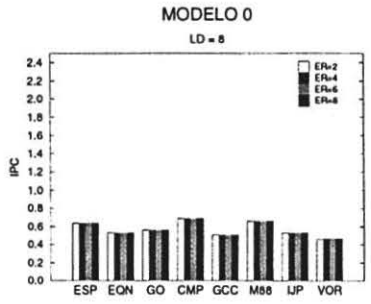
TABELA I
LIMITES SUPERIORES DE DESEMPENHO

| | Menor Valor | Maior Valor |
|---|---|---|
| IPC | 1.10 | 2.0 |
| IPCmem | 0.28 | 0.82 |
| Bloqueio | 0% | 33% |

A Figura 1 mostra o IPC alcançado pela arquitetura super escalar que utiliza o Modelo 0. Nesta figura aparece apenas o gráfico para configurações com largura de despacho de 8 instruções. Os desempenhos das configurações com larguras de despacho de 2, 4 e 6 instruções são essencialmente idênticos ao obtido na configurações com largura de despacho de 8 instruções, e por isso os gráficos correspondentes não serão aqui apresentados.

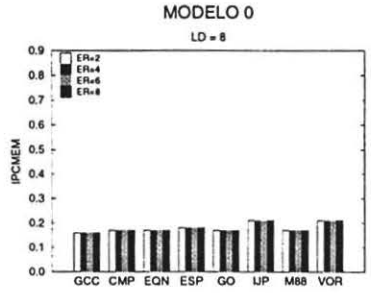
Estes valores de desempenho refletem a ineficiência do Modelo 0. A Figura 2 mostra o IPCmem de Modelo 0, também para configurações com larguras de despacho de 8 instruções (os valores para larguras de despacho de 2, 4 e 6 instruções são idênticos).

A alta proporção de instruções de memória e a baixa vazão do Modelo 0 produzem dois efeitos: (1) são geradas cadeias de dependências longas, devido à latência entre o despacho e a execução de instruções LOAD; (2) ocorre uma alta taxa de ocupação de recursos devido ao acúmulo das instruções de acesso à memória. Este último efeito é particularmente acentuado, conforme observado nos gráficos da Figura 3, que mostram o perfil do bloqueio do despacho para o Modelo 0.



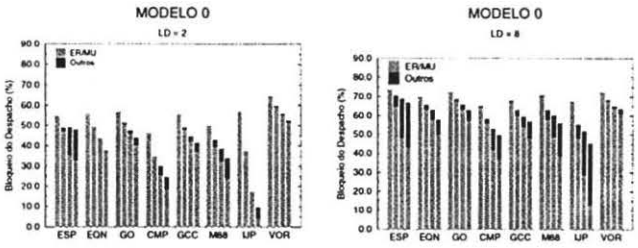

Largura de despacho = 8.

Fig. 1. IPC com o Modelo 0.



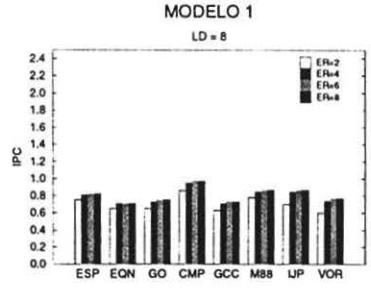

Largura de despacho = 8.

Fig. 2. IPCmem com o Modelo 0.




Largura de despacho = 2. Largura de despacho = 8.

Fig. 3. Bloqueio do despacho com o Modelo 0.



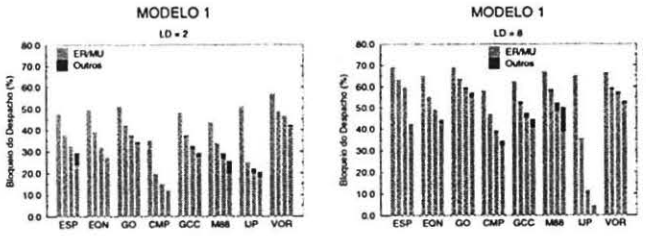

Largura de despacho = 8.

Fig. 4. IPC com o Modelo 1.

A Figura 4 mostra o IPC alcançado pela arquitetura que usa o Modelo 1, para largura de despacho de 8 instruções. Da mesma forma que no Modelo 0, não há diferença significativa de desempenho com a variação da largura de despacho. Os níveis de desempenho para as larguras de despacho extremas (2 e 8) são essencialmente idênticos. No entanto, ao contrário do que acontece no Modelo 0, o desempenho da arquitetura passa a ser sensível ao número de estações de reserva da unidade de memória. O IPC médio para configurações com largura de despacho de 8 instruções e com 2 estações de reserva é 0.70 instruções/ciclo, sendo 0.80 instruções/ciclo para configurações com 6 ou 8 estações de reserva, ou seja, uma diferença porcentual de 15%.

Com relação à taxa de execução de instruções de memória no Modelo 1, o maior IPCmem foi 0.40 instruções de memória por ciclo para a configuração com largura de despacho de 8 instruções e com 8 estações de reserva para o programa `vortex`. Para esta mesma configuração, o IPCmem médio ao longo de todos os programas foi 0.31 instruções de memória por ciclo, o que representa um ganho de 75% em relação ao Modelo 0.

Os gráficos na Figura 5 mostram o perfil do bloqueio do despacho em função da indisponibilidade de estações de reserva de memória. Observa-se uma importante redução nas taxas de bloqueio quando comparadas ao Modelo 0.



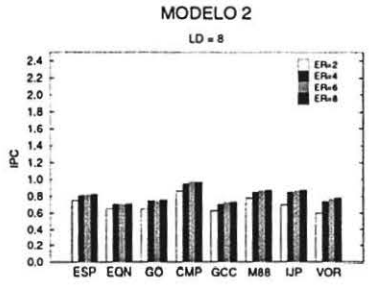

Largura de despacho = 2. Largura de despacho = 8.

Fig. 5. Bloqueio do despacho com o Modelo 1.

O Modelo 1 introduziu ganhos de desempenho evidentes no IPC e IPCmem. Embora as taxas de bloqueio tenham diminuído em relação ao Modelo 0, estas continuam muito elevadas quando comparadas àquelas no Modelo Ideal. São dois os fatores que contribuem para estes níveis de bloqueio do despacho no Modelo 1: (1) o *pipeline* é bloqueado nos casos de *miss*; e (2) instruções de memória especulativas não são escalonadas para execução. O primeiro fator não tem um peso muito grande, devido à baixa taxa de *miss* (taxa de *hit* acima de 95%). O segundo fator é mais importante, pois ele restringe a exploração do paralelismo entre as instruções de memória, aos limites de um bloco básico.

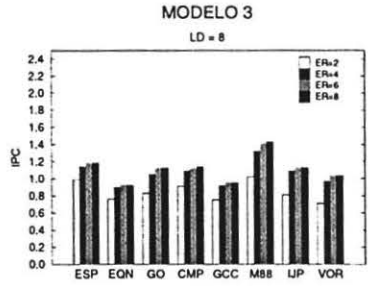
A Figura 6 mostra, o IPC para o Modelo 2. Observamos que os níveis de desempenho dos Modelos 1 e 2 são idênticos. Esta similaridade deve-se à baixa taxa de *miss* observada para os programas de teste, fazendo com que a introdução de acessos fora de ordem e de *forwarding* não tragam um acréscimo significativo de desempenho. Os níveis de bloqueio do Modelo 2 e a taxa de execução de instruções de memória são idênticos aos do Modelo 1.

As seguintes medidas adicionais permitem um melhor entendimento da dinâmica de funcionamento do Modelo 2. Em qualquer configuração, a MAQ permanece cerca de 99% dos ciclos vazia. Nos ciclos restantes ou seja, 1%, com apenas uma posição ocupada, sendo que raramente (0.01%) duas posições são ocupadas na MAQ. O porcentual de instruções que efetivamente acessam a memória *cache* fora de ordem é de 0.22%, 0.3%, 0.31% e 0.31% para 2, 4, 6 e 8 estaçoes de reserva respectivamente, para configurações com largura de despacho de 4 instruções. No Modelo 2, a taxa de repasse de dados entre instruções dependentes (*forwarding*) é praticamente nula.




Largura de despacho = 8.

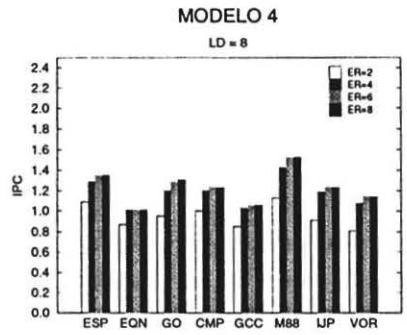
Fig. 6. IPC com o Modelo 2.




Largura de despacho = 8.

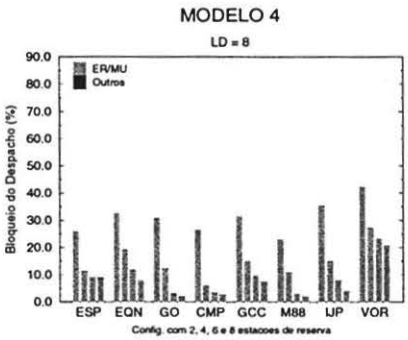
Fig. 7. IPC com Modelo 3.

A Figura 7 mostra o IPC alcançado pela arquitetura que usa o Modelo 3 de acesso à memória. Este modelo apresentou um IPC médio 36% superior ao do Modelo 2 (todos os valores apresentados deste ponto em diante são para a configuração com largura de despacho de 8 instruções e com 8 estações de reserva de memória). Para a mesma configuração, o IPCmem médio é 25% superior ao do Modelo 2, enquanto a taxa média de bloqueio é 65% menor.




Largura de despacho = 8.

Fig. 8. IPC com o Modelo 4.




Largura de despacho = 8.

Fig. 9. Bloqueio do despacho com o Modelo 4.

Medidas adicionais mostram que as facilidades herdadas de modelos anteriores são melhor aproveitadas no Modelo 3. O porcentual de instruções de memória executadas fora de ordem aumenta de 0.005% no Modelo 2 para 13% no Modelo 3, no programa `m88ksim`. A única exceção ocorreu com o programa `espresso`, cujo percentual de acessos fora de ordem diminuiu de 0.041% no Modelo 2 para 0.039% no Modelo 2 (muito baixo e praticamente inalterado). Entretanto, para este programa o Modelo 3 trouxe um aumento no IPC de 44%. O porcentual de instruções completadas especulativamente para o programa `m88ksim` foi de 58.6%, ou seja, mais da metade do total de acessos à cache. Esses dados explicam o ótimo ganho de 37% no desempenho do Modelo 3 em relação ao Modelo 2.

A Figura 8 mostra o desempenho alcançado pela arquitetura que usa o Modelo 4 de acesso à memória. Comparando seu desempenho médio com o Modelo 3, verifica-se que houve um ganho de 10,8%. O maior desempenho se verificou com o programa `m88ksim` com IPC de 1.53. O menor IPC foi de 1.01 para o programa `eqntott`.

Observando as taxas de bloqueio na Figura 9, nota-se uma redução considerável do bloqueio em relação ao Modelo 3, sendo em média, superior a 70%.

## VI. Conclusões

Neste trabalho foram avaliados os efeitos de diversos modelos de acesso à memória sobre o desempenho de uma arquitetura super escalar típica. Para realizar esta tarefa, foram implementados 5 modelos de acesso. O comportamento de cada modelo foi monitorado através de diversas métricas de desempenho.

Analisando os gráficos de desempenho e os dados adicionais apresentados na seção anterior, verifica-se que ganhos significativos de desempenho foram obtidos principalmente com dois modelos: o Modelo 1, que introduziu execução em *pipeline* e resultou em um ganho médio de 41% no desempenho em relação ao Modelo 0; e o Modelo 3, que resultou em um ganho médio de 37% em relação ao Modelo 1 e de 94% em relação ao Modelo 0.

O Modelo 2 não apresentou ganho em relação ao Modelo 1 em virtude das modificções introduzidas visarem reduzir a penalidade de *miss*, sendo esta muito baixa na configuração de memória adotada. Entretanto, as facilidades introduzidas servem de suporte para os modelos especulativos.

Quanto ao Modelo 4, observou-se um ganho de 113% em relação ao Modelo 0, de 51% em relação ao Modelo 1 e um ganho de 10% em relação ao Modelo 3. A diferença em relação ao Modelo 3 pode parecer pouco expressiva, devido à agressividade do Modelo 4. No entanto, é necessário dispor de um melhor contexto para julgar a importância de ganhos desta ordem. Por exemplo, deve-se conhecer (1) o custo de implementação do mecanismo e (2) a faixa de desempenho alvo da arquitetura. Estes dois elementos adicionais permitem um julgamento mais apropriado baseado na relação custo-benefício. Caso a arquitetura seja destinada a aplicações de baixo/médio desempenho, um ganho de 10% pode ser considerado marginal tendo em vista a provável complexidade de implementação do Modelo 4. No entanto, para faixas de aplicações que requerem o maior desempenho possível, o ganho de 10% introduzido com o Modelo 4 pode valer plenamente o seu custo de implementação.

## References

[1] Patterson, D., Hennessy, J., *Computer Architecture: A Quantitative Approach*, 2nd. Ed., Morgan-Kaufmann, San Francisco, CA, 1997.

[2] Diep, T. A., Shen, J. P., *Performance Evaluation of the PowerPC 620 Microarchitecture*, Proceedings of the 22nd International Symposium on Computer Architecture, 1995, pp. 163–175.

[3] Franklin, M., Sohi, G. S., *ARB: A Hardware Mechanism for Dynamic Memory Disambiguation*, IEEE Transactions on Computers, Vol. 45, No. 5, May 1996, pp. 552–571.

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

[5] Systems Performance Evaluation Corporation, *SPEC95 User Manual*, 1995.