Escalonamento adaptativo ao uso da hierarquia de memória para máquinas multiprocessadas

Maurício Pillon* e Olivier Richard
Laboratório de Informática e Distribuição
Projeto APACHE
Grenoble, França
{Mauricio.Pillon,Olivier.Richard}@imag.fr

Resumo

A evolução da tecnologia empregada na fabricação das memórias é mais lenta do que as empregadas aos processadores. O acesso intensivo à hierarquia de memória neste tipo de máquina, provoca a queda do desempenho das aplicações. A monitoração das transações no barramento de memória permitiu estabelecer uma relação entre a taxa de acesso a este barramento e ao desempenho das aplicações. Esta monitoração foi feita através dos contadores de desempenho em hardware. *Graças a esta relação tornou-se possível estimar o* speed-up *de uma aplicação durante a execução.*

*Neste contexto, o projeto DRAC (*aDaptive contRol system with hArdware performance Counters*) propõe um sistema de controle adaptável que visa maximizar a utilização dos recursos baseado no relacionamento entre o uso da memória e o desempenho das aplicações. A estratégia de escalonamento de DRAC busca evitar a saturação no barramento de memória, permitindo o aumento de desempenho.*

Este artigo descreve o protótipo do sistema DRAC através de um estudo do relacionamento entre a utilização da hierarquia de memória e o speed-up *em máquinas quadri-processadas.*

1. Introdução

O impacto da hierarquia de memória em máquinas multiprocessadas é conhecido na comunidade científica [7]. A freqüência dos componentes da hierarquia de memória é entre 2 à 8 vezes mais lenta que os processadores e, devido a esta diferença, a hierarquia de memória é, seguidamente, um dos maiores gargalos de desempenho das aplicações. A monitoração dos acessos ao barramento de memória, permitiu o estudo detalhado deste gargalo. Esta monitoração foi possível graças a disponibilização dos eventos de memória (*cache misses*, *cache hits* ou *bus memory transactions*) pelos contadores de desempenho em *hardware*.

Trabalhos como Polychronopoulos [10] e Cappello [2] mostraram que o uso intensivo da hierarquia de memória provoca a queda de desempenho das aplicações. O trabalho apresentado neste artigo desenvolveu um estudo aprofundado sobre o uso da hierarquia de memória, o qual identificou uma relação entre a queda de desempenho e a taxa de utilização do barramento de memória. Esta relação permite estimar o *speed-up* da aplicação e identificar o gargalo de memória durante a execução.

A metodologia de observação das aplicações, já é utilizada há muito tempo para a identificação de gargalos, e uma das técnicas de observação mais usadas é a monitoração. Projetos como Magnet [5], CODE [14] e Remos [4] propõem bibliotecas para a monitoração de redes, enquanto Paradyn [9] disponibiliza um ambiente de observação genérico para a identificação de gargalos e possui módulos de observação configuráveis dinamicamente.

A monitoração durante a execução, permitiu o desenvolvimento de sistemas de controle adaptável. Autopilot [13], por exemplo, faz o controle adaptável de sistemas paralelos e distribuídos, onde a monitoração é feita pelo sensores (*sensor*) e as modificações pelos atores (*actors*). A tomada de decisão, tanto local quanto global, é baseada em lógica difusa. O sistema desenvolvido pela equipe coordenada por Polychronopoulos [10] faz o controle de processos adaptável conforme as requisições à memória auxiliar (*swap*) com o objetivo de reduzir a transferência dos dados entre o disco e a memória principal.

* Bolsista de Doutorado GDE número 200 242/00-8 - CNPq - Brasil

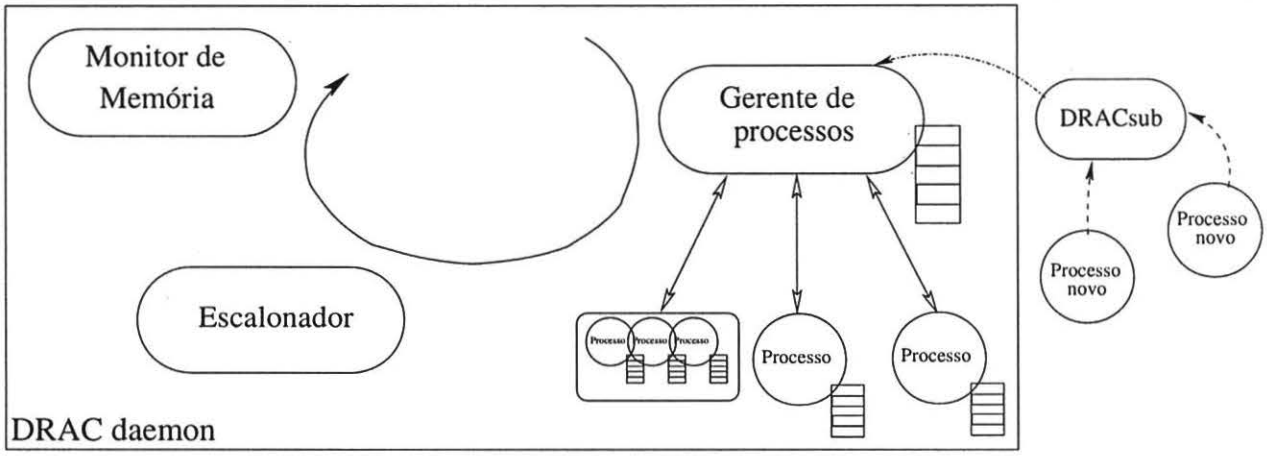

Figura 1. Sistema de controle adaptável: O protótipo do sistema DRAC é composto de duas aplicações a nível de usuário. *DRACsub*, que se encarrega do lançamento e controle de submissão de processos, e *DRAC daemon*, responsável pelo escalonamento e gerenciamento de processos e pela monitoração da memória. O *daemon* é um loop seqüencial infinito que possui uma função para cada um de seus módulos.

A grande maioria dos sistemas de monitoração existentes levam em consideração somente as atividades efetuadas na camada sistema ou na camada usuário, no entanto, a observação das atividades efetuadas em *hardware* podem facilitar a identificação de problemas como a contensão no barramento de memória. Os contadores de desempenho em *hardware* tornaram-se populares após o surgimento das bibliotecas de interface, como Vtune [6], PAPI [8], HPM [3] e PCL [1]. O objetivo comum destas bibliotecas é facilitar a utilização destes contadores de desempenho. A biblioteca Vtune suporta os contadores das arquiteturas Intel (Pentium, Itanium I e itanium II) e a HPM das arquiteturas IBM (PowerPC, Power3 e Power4). Cada processador possui um formato específico de contador de desempenho e uma quantidade de eventos diferentes, tornando complexa a portabilidade dos programas. Neste contexto, sugiram as bibliotecas PCL e PAPI, as quais disponibilizaram uma interface (API) padrão de acesso aos contadores e definiram um conjunto de eventos comuns a todos estes processadores.

DRAC é um sistema de controle adaptável que faz o gerenciamento de processos baseado na utilização de memória. Através dos contadores de desempenho em *hardware*, DRAC acompanha a evolução da utilização do barramento de memória durante a execução. Assim que o monitor de memória de DRAC identifica a formação de gargalo neste barramento, o escalonador adaptativo efetua um novo escalonamento afim de evitar a saturação do recurso de memória.

Este artigo está organizado da seguinte forma: a seção 2 apresenta o protótipo do sistema DRAC; a seção 3 expõe os resultados sobre o impacto da utilização da hierarquia de memória nos desempenhos das aplicações em máquinas quadri-processadas; o modelo matemático destas máquinas quadri-processadas será apresentado na seção 4 e finalmente, os resultados iniciais das diferentes estratégias de escalonamento serão mostrados na seção 5.

2. Sistema de controle adaptável

O sistema de controle adaptável, DRAC [12] (*aDaptive contRol system with hArdware performance Counters*), visa maximizar a utilização dos recursos de máquinas multiprocessadas, baseando-se na observação das atividades sobre a hierarquia de memória. A contensão na hierarquia de memória, devido a saturação no barramento, implica na queda de desempenho. A arquitetura DRAC propõem o escalonamento adaptativo de processos de acordo com a utilização da hierarquia de memória.

Esta seção tem como principal objetivo apresentar os componentes do protótipo do sistema de controle adaptável, DRAC. Este sistema (Figura 1) possui duas aplicações: uma de submissão de processos, o *DRACsub* e outra de controle de processos, o *DRAC daemon*. Ambas as aplicações encontram-se na camada usuário.

A aplicação *DRACsub* permite o lançamento dos processos pelo usuário. O número de processos em execução no sistema é limitado pela quantidade de memória disponível,

e cabe a *DRACsub* a verificação destes limites e o armazenamento, se necessário, das requisições pendentes.

A segunda aplicação do sistema DRAC, o *DRAC daemon*, é composta por três módulos:

- **Monitor de memória:** é responsável pelo monitoramento da evolução da utilização de memória. A observação do recurso de memória é efetuada através dos contadores de desempenho em *hardware*, no qual o evento de *hardware* escolhido permite ao monitor a contagem de todas as transações no barramento de memória.

- **Escalonador:** baseia-se nas informações recolhidas pelo monitor de memória para a tomada de decisão. Afim de evitar os escalonamentos desnecessários, este escalonador considera dois limites para a tomada de decisão: um limite inferior que marca a área de sub-utilização e, um limite superior que marca a área de início da saturação do barramento. Enquanto a utilização no barramento de memória fica entre estes dois limites, o escalonador de DRAC não é acionado. Assim que a média da utilização de memória entre as últimas amostras ultrapassar o limite superior, o escalonador identifica o processo que fez o maior número de acessos ao barramento neste último período e encaminha o pedido de substituição deste processo. O mesmo procedimento é feito caso a utilização seja menor que o limite inferior, no entanto, desta vez, o escalonador requisitará a substituição do processo com menor número de acesso ao barramento.

- **Gerente de processos:** é responsável pela inclusão, remoção, substituição e sincronização dos processos. A substituição dos processos são baseadas no histórico de execução do mesmo, caso ele exista. A sincronização é possível graças a interceptação das chamadas de funções pelo sistema. Esta interceptação é feita de duas formas: através da alteração do binário do compilador ou através da modificação de algumas funções das bibliotecas de programação. Atualmente, o sistema suporta aplicações multithreads e OpenMP [15] (se compilado com PGI).

O protótipo apresentado nesta seção baseou-se na arquitetura genérica de DRAC [12]. As próximas seções são dedicadas à descrição da hierarquia de memória através de um modelo matemático e à avaliação deste protótipo para máquinas quadri-processadas.

3. Desempenho X utilização da memória

Esta seção tem por objetivo apresentar a relação entre a queda de desempenho e a utilização de memória em máquinas quadri-processadas. Esta relação permite detectar a eficiência na utilização da memória durante a execução.

Os testes foram efetuados nas seguintes máquinas quadri-processadas: Pentium Pro $200MHz$ com uma freqüência do barramento de memória de $66MHz$ e com $256KB$ de memória cache L2 e Pentium III $550MHz$ com uma freqüência do barramento de memória de $100MHz$ e com $256KB$ de memória cache L2.

Os experimentos consistem em três cálculos de vetores, cujas as funções escolhidas foram *cópia*, *stride* e *indireção*. A primeira função faz a cópia seqüencial de um vetor A a um outro vetor B. A segunda função faz a cópia de um vetor A a um outro vetor B provocando *cache misses* para cada novo elemento. A função *indireção* faz a cópia entre vetores indexado por um terceiro vetor. Todos os vetores são inicializados com dados aleatórios, ocupam em torno de 80% da memória total e são do tipo flotante. A fase de inicialização foi excluída das medidas.

Estas funções visam mostrar a evolução do *speed-up* das aplicações de acordo com a taxa de utilização do barramento de memória. A variação do *speed-up* foi feita através do acréscimo de instruções assembler (*nop*). A instrução *nop* ocupa o processador por um ciclo de relógio e, desta forma, o número total de requisições à hierarquia de memória não é modificado, mas no entanto, a intensidade de requisições por segundo é reduzida. O número de transações por segundo é obtido através dos contadores de desempenho em *hardware*, o que representa a taxa de utilização do barramento de memória. A biblioteca utilizada para a leitura destes contadores foi Perfctr [11], *patch* do *kernel* desenvolvido por Mikael Pettersson. Perfctr faz parte da camada de base de PAPI para as arquiteturas Intel e AMD 32bits. O evento escolhido conta as transações efetuadas no barramento de memória (*BUS_TRAN_MEM*).

As Figuras 2 e 3, apresentam os resultados obtidos nas duas máquinas quadri-processadas. Em ambas, o *speed-up* é próximo ao ótimo quando a taxa de utilização do barramento de memória é baixa. A medida em que a taxa de utilização do barramento de memória aumenta, o *speed-up* diminui, chegando próximo de 1 para o computador Pentium Pro e, a menos de 2 para o Pentium III. Na Figura 2, observa-se a leve queda do *speed-up* até que a taxa de utilização do barramento de memória atinja $4.5x10^6$, e a partir deste ponto o *speed-up* cai bruscamente, de pouco mais de 3 para próximo de 1.

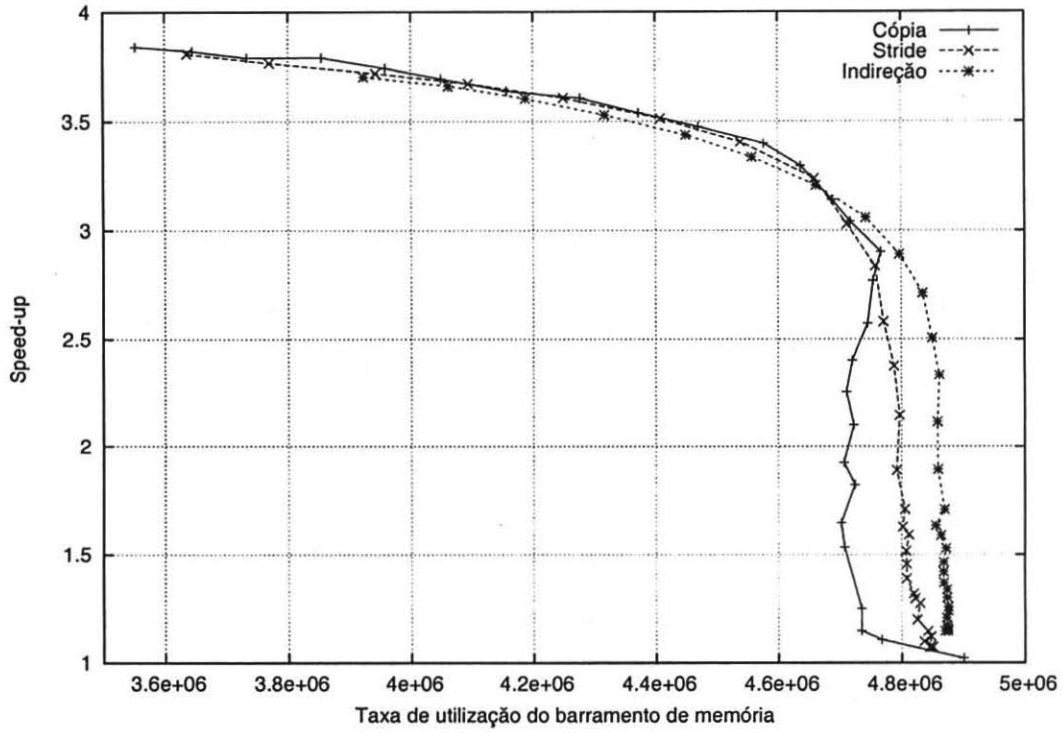

Figura 2. Evolução da taxa de utilização do barramento de memória de acordo com o *speed-up*. O número de transições por segundo no barramento de memória é medido através dos contadores de desempenho em *hardware*. Esta máquina (Pentium Pro quadri-processado) apresenta uma quebra brusca de desempenho assim que a taxa de utilização do barramento de memória ultrapassa as $4.6x10^6$ transações por segundo.

O comportamento obtido para o Pentium III não é muito diferente, no entanto, como mostra a Figura 3, onde pode-se constatar que a queda do *speed-up* passou por uma etapa intermediária. Num primeiro momento, o *speed-up* teve uma leve queda chegando próximo de 3 para todas as taxas de memória inferiores a $1.4x10^7$. No período seguinte, onde as taxas de memória ficaram entre $1.4x10^7$ e $1.5x10^7$, o *speed-up* ficou entre 2.5 e 3, e, finalmente, assim que as taxas de memória ultrapassam $1.5x10^7$ os *speed-up* ficam em torno de 2, em certo casos caem a quase 1. Estes experimentos comprovaram, como já era esperado, que a contensão de memória em máquinas quadri-processadas é maior que em máquinas biprocessadas [12].

A seção 4 apresenta a extensão do modelo matemático para máquinas quadri-processadas, o que permite avaliar os ganhos de desempenho nestas máquinas.

4. Modelo matemático

As seções precedentes apresentaram o sistema de controle adaptável, DRAC, e o estudo sobre o impacto da saturação da hierarquia de memória no desempenho das aplicações. Esta seção tem por objetivo a descrição formal do problema de contensão da hierarquia de memória. O modelo proposto calcula o tempo de execução de um grupo de processos de acordo com três tipos de escalonamentos.

O principal objetivo deste modelo é o de avaliar a qualidade e o *overhead* do escalonador implementado no protótipo do sistema DRAC (seção 2).

Hipóteses:

1. Um sistema de memória simples: um barramento de memória compartilhado por todos os processadores.
2. Todos os n processos em execução possuem uma única dependência, o acesso concorrente ao barramento de memória.
3. O fluxo de instrução de um processo é o mesmo se executado em seqüencial exclusivo ou em concorrência com outros processos.
4. O uso da memória de um processo é constante durante todo o seu tempo de execução.

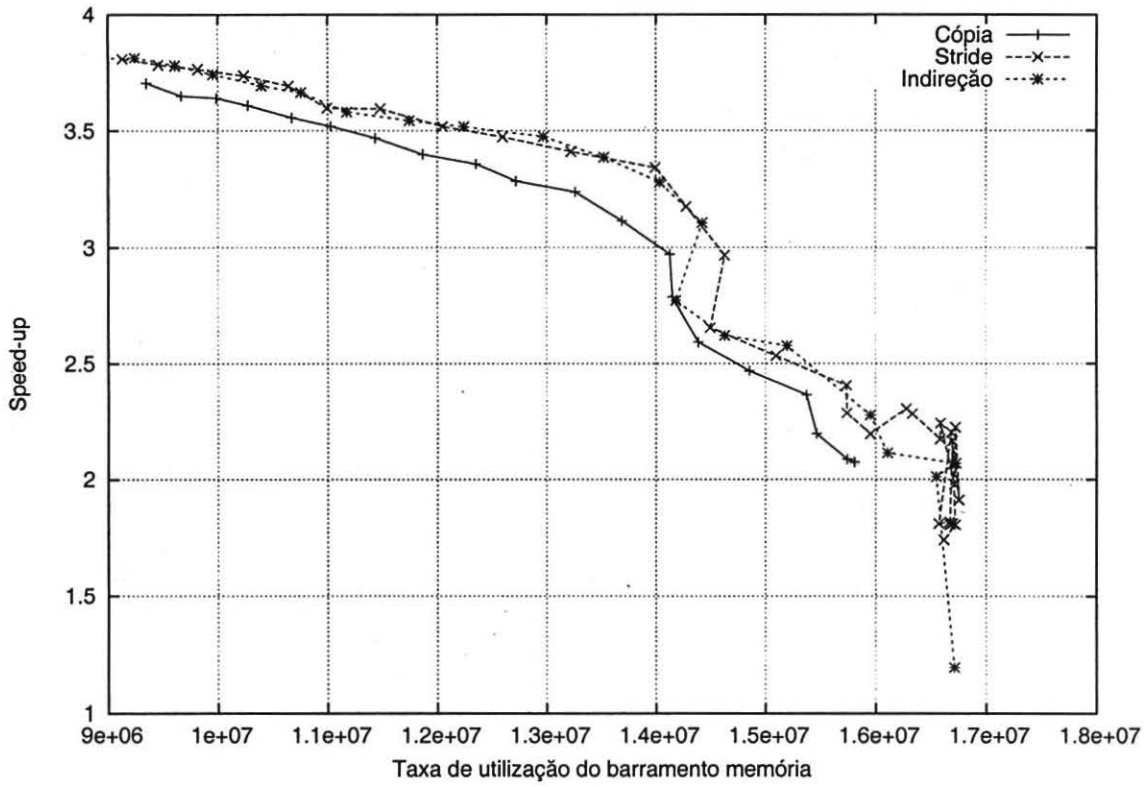

Figura 3. Evolução da taxa de utilização do barramento de memória de acordo com o *speed-up*. O número de transições por segundo no barramento de memória é medido através dos contadores de desempenho em *hardware*. A queda do *speed-up* nesta máquina (Pentium III quadri-processado) apresenta dois estágios, um primeiro ao passar os $1.4x10^7$ e um segundo ao atingir os $1.5x10^7$.

Definições: O processo exclusivo é um processo seqüencial que executa-se sozinho na máquina, enquanto o processo concorrente é aquele que executa-se em concorrência com outros. n é o número de processos. Os processos executam-se em p processadores. $d_{seq}(i)$ é a taxa de utilização do barramento de memória pelo processo i para $i \in \{1 \ldots n\}$. d_{max} é taxa máxima de utilização suportada pelo barramento de memória. Existem três tipos de cargas de memória:

- A carga de memória de um processo i: é $\alpha(i) = \frac{d_{seq}(i)}{d_{max}}$, onde α representa a proporção utilizada do barramento de memória pelo processo exclusivo i. Se $\alpha(i) = 0$, a taxa de acesso ao barramento de memória do processo i é irrelevante. Se $\alpha(i) = 1$, a taxa de acesso ao barramento de memória do processo i atinge a capacidade máxima do barramento de memória.
- Carga teórica da máquina: $\frac{\sum_{i=1}^{n} d_{seq}(i)}{d_{max}}$, a carga teórica da máquina pode ser maior que 1, caso a soma das taxas de acesso ao barramento de memória dos processos em execução seja maior do que a capacidade máxima do barramento de memória.
- Carga real da máquina: $min(1, \frac{\sum_{i=1}^{n} d_{seq}(i)}{d_{max}})$, onde a carga real é 1 para todo valor maior que 1 ou igual à carga teórica para os demais valores.

O tempo total de execução de um grupo de processos:

- caso haja saturação do barramento de memória ($\sum_{i=1}^{n} d_{seq}(i) > d_{max}$) é:

$$\boxed{T_{all} = \frac{t_{seq}}{p} * max(1, \frac{\sum_{i=1}^{n} d_{seq}(i)}{d_{max}})} \quad (1)$$

  onde o tempo seqüencial exclusivo t_{seq} representa o tempo gasto para executar n processos, um após o outro ($t_{seq} = \sum_{i=1}^{n} t_i$).
- caso contrário ($\sum_{i=1}^{n} d_{seq}(i) < d_{max}$) é:

$$\boxed{T_{all} = \frac{t_{seq}}{p}} \quad (2)$$

4.1. Estudo de Caso: quadri-processador

Nesta seção será enfatizado o estudo do modelo descrito na seção precedente para máquinas quadri-processadas. O estudo de caso em máquinas duo-processadas [12] comprovou a relevância deste modelo. Os três tipos de escalonamentos modelados são: T_{min} *Melhor Escalonamento*, T_{max} *Pior Escalonamento* e T_{rand} *Escalonamento Aleatório*. Em todos os casos serão avaliados o tempo total de execução T_{all}.

O cenário é composto por dois tipos de processos P_0 e P_1. A taxa de acesso ao barramento de memória de P_0 é irrelevante ($\alpha(P_0) = 0$) e a taxa de acesso ao barramento de memória de P_1 é próxima a capacidade total deste barramento ($\alpha(P_1) \in [0.9, 1]$). O *speed-up* de um processo é t_{seq}/t_{all}, para a execução de quatro processos do tipo P. O *speed-up* de P_0 é quatro e o *speed-up* de P_1 é δ ($\delta = \frac{4}{max(1,4\alpha)}$). β é a porcentagem de processos P_1, ou seja, $\beta = \frac{n_{P_1}}{N}$ onde N é o número total de processos: $N = n_{P_0} + n_{P_1}$. O tempo de execução concorrente de P_i e P_j é $t_p(k.P_i, (p-k).P_j)$, onde $(i,j) \in \{0 \ldots 1\}^2$ e $k \in \{0 \ldots p\}$. Os tempos de execução seqüencial exclusivo de P_0 e P_1 são iguais.

Então, o tempo total de execução (T_{all}) do grupo de processos é dado pela fórmula abaixo. Os processos executam-se em grupos de 4, sabendo que, P_1 possui $\alpha(P_1)$ entre 0.9 e 1 e o escalonamento de dois ou mais processos P_1 provocam a saturação do barramento de memória. O fator de redução de desempenho provocado pela contensão neste barramento é de $4/\delta$. $T_{k.P_i(p-k).P_j}$ representa o tempo total de execução de todos os segmentos onde $k.P_i$ executa-se em concorrência com $(p-k).P_j$ para um k específico e $t_{th}(k.P_i(p-k).P_j)$ representa o mesmo grupo de processos em execução, sem considerar os atrasos provocados pela contensão do barramento de memória.

$$T_{all} = t_{th}(4.P_0 0.P_1) + t_{th}(3.P_0 1.P_1) + \frac{2}{\delta} t_{th}(2.P_0 2.P_1) + \frac{3}{\delta} t_{th}(1.P_0 3.P_1) + \frac{4}{\delta} t_{th}(0.P_0 4.P_1)$$

O cálculo do tempo total de execução para o *Escalonamento Aleatório* foi baseado na probabilidade de execução de cada grupo de processos. As probabilidades são:

$$Prob(4.P_0 0.P_1) = (1-\beta)^4$$
$$Prob(3.P_0 1.P_1) = 4.\beta(1-\beta)^3$$
$$Prob(2.P_0 2.P_1) = 6.\beta^2(1-\beta)^2$$
$$Prob(1.P_0 3.P_1) = 4.\beta^3(1-\beta)$$
$$Prob(0.P_0 4.P_1) = \beta^4$$

assim, a fórmula obtida é:

$$T_{rand} = \frac{N}{4}[(\beta^4) + (4.\beta(1-\beta)^3) + \frac{2}{\delta}(6.\beta^2(1-\beta)^2) + \frac{3}{\delta}(4.\beta^3(1-\beta)) + \frac{4}{\delta}((1-\beta)^4)]$$

e após o desenvolvimento:

$$T_{rand} = \frac{\beta^2 N}{\delta'}(\beta^2 - 3\cdot\beta + 3) + \frac{N}{4}(-3\cdot\beta^4 + 8\cdot\beta^3 - 6\cdot\beta^2 + 1)$$

O *Melhor Escalonamento* é obtido através da minimização do tempo onde P_1 executa-se em concorrência com outros processos deste mesmo tipo. Para $\beta \leq 25\%$, o escalonador lança três processos P_0 contra um processo P_1 até que todos os P_1 se acabem, finalizando a execução deste grupo com o restante de processos P_0, caso ainda haja. Desta forma, o escalonador evita a contensão do barramento de memória durante toda a execução. Para $25\% \leq \beta \leq 50\%$, o escalonamento ótimo é o lançamento de dois processos P_0 contra dois outros processos P_1 até que $\beta - 25\% \leq 0$. A partir deste ponto, a regra aplicada será a mesma que para $\beta \leq 25\%$. A execução de P_1 em concorrência com P_1 é inevitável. O atraso causado pela contensão neste período é de $\frac{2}{\delta}$. Para $\beta \geq 50\%$, o escalonador lança dois processos P_0 contra dois outros P_1, finalizando por quatro processos P_1.

A representação formal para o *Melhor Escalonamento* é:

- Se $0\% \leq \beta \leq 25\%$:

$$T_{min} = \beta N + \frac{(1-\beta)N - 3\cdot\beta N}{4}$$
$$T_{min} = \frac{N}{4}$$

- Se $25\% \leq \beta \leq 50\%$:

$$T_{min} = \frac{2N}{\delta}\cdot(\beta - \frac{1}{4}) + (4\cdot(\frac{1}{2} - \beta)\cdot\frac{N}{4})$$
$$T_{min} = \frac{\beta N}{\delta}$$

- Se $50\% \leq \beta \leq 100\%$:

$$T_{min} = \frac{2}{\delta}\cdot\frac{(1-\beta)N}{2} + \frac{\frac{4}{\delta}\cdot(\beta N - (1-\beta)N)}{4}$$
$$T_{min} = \frac{\beta N}{\delta}$$

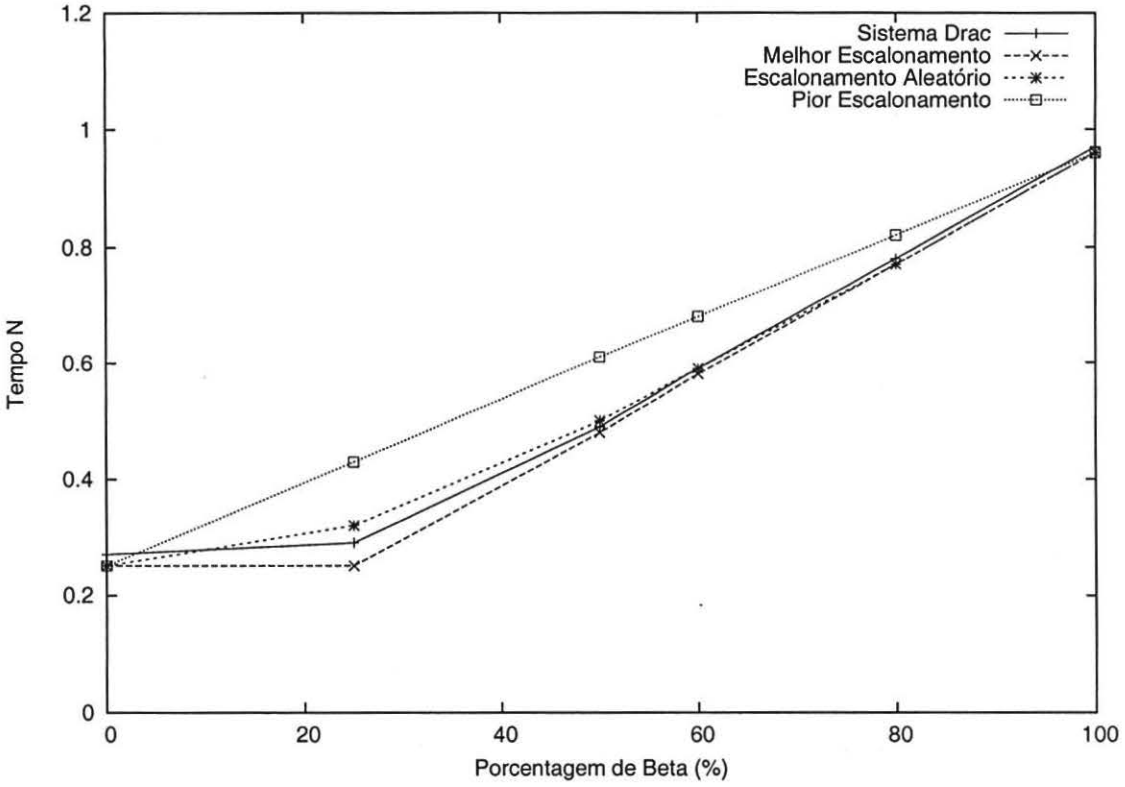

Figura 4. No eixo x encontram-se as porcentagens β de processos, e no eixo y os tempos de execuções N normalizado. Os resultados representam os tempos obtidos por cada estratégia de escalonamento de acordo com a porcentagem β de processos. A única estratégia medida é a do sistema DRAC, as outras três foram obtidas através do modelo matemático.

O *Pior Escalonamento* é obtido através da maximização do tempo de execução dos processos P_1 em concorrência com outros processos P_1.

$$T_{max} = \frac{N}{4} \cdot (1 - \beta) + \frac{N}{4} \cdot \frac{4}{\delta} \cdot \beta$$
$$T_{max} = \frac{N}{4} \cdot ((1 - \beta) + \frac{4}{\delta} \cdot \beta)$$

De acordo com o modelo apresentado, a maior variação entre os tempos de execução é para $\beta = 25\%$. Neste caso, o *Melhor Escalonamento* é 75% mais rápido que o *Pior Escalonamento*. A próxima seção mostra os ganhos obtidos pelo escalonador do sistema DRAC, conforme o modelo.

5. Resultados Experimentais

Estes resultados experimentais provêm da comparação dos tempos obtidos através do modelo matemático e dos tempos medidos através do protótipo do sistema DRAC. Os testes foram constituídos por dois tipos de processos, um com $\alpha = 0$ e outro com $\alpha = 0.9$. Os processos com $\alpha = 0$ tem em média uma taxa de acesso ao barramento de memória em torno de 0.1% da capacidade deste barramento. Os processos com $\alpha = 0.9$ ocupam mais de 90% deste barramento.

As estratégias de escalonamento são: o *Melhor Escalonamento*, o *Pior Escalonamento* e o *Escalonamento Aleatório* do modelo matemático e o *Escalonamento DRAC* medido. A Figura 4 apresenta no eixo x a porcentagem (β) de processos do tipo $\alpha = 0.9$ com os valores $0\%, 25\%, 50\%, 60\%, 80\% e 100\%$ e no eixo y os tempos de execução normalizados.

A máquina utilizada para os testes foi o quadri-processado Pentium Pro[1]. Os resultados mostram que o *Escalonamento DRAC* tem melhor desempenho que o *Escalonamento Aleatório* para todos os $\beta > 10\%$. O *Melhor Escalonamento* chega a 22% mais rápido que o *Escalonamento Aleatório*. O melhor desempenho obtido pelo *Esca-*

1 Os testes não puderam ser lançados na máquina quadri-processada Pentium III devido a problemas de disponibilidade da mesma.

lonamento DRAC em relação ao *Escalonamento Aleatório* foi de 10%.

Os primeiros resultados em máquinas quadri-processadas são promissores. A redução das trocas de contextos deve diminuir o *overhead* para $\beta < 10\%$ e aumentar os ganhos.

6. Conclusão e trabalhos futuros

Este trabalho apresentou a arquitetura de DRAC, um sistema de controle adaptável, que busca maximizar a utilização dos recursos de máquinas multiprocessadas. A observação das atividades dos componentes da hierarquia de memória permitiu a identificação de gargalos de desempenho. O estudo aprofundado destas atividades provou que o aumento gradativo dos acessos ao barramento de memória provoca a perda de desempenho e, que a monitoração das atividades efetuadas no barramento de memória torna possível estimar o *speed-up* das aplicações.

Os experimentos feitos em máquinas quadri-processadas mostraram que a saturação da hierarquia de memória provocam a perda de mais de 50% do desempenho das aplicações. De acordo com os tempos calculados através do modelo, o *Pior Escalonamento* é 50% mais lento que o *Melhor Escalonamento*. O escalonador implementado no protótipo de DRAC obteve um ganho de até 10% em relação ao *Escalonamento Aleatório*, no entanto, o uso do protótipo mostrou-se limitado para $\beta < 10\%$. O problema de *overhead* neste caso, pode ser solucionado através da adaptação dos intervalos de tempo para a tomada de decisão.

O trabalho apresentado obteve resultados importantes nas máquinas quadri-processadas e permitiu a elaboração das próximas etapas. A curto prazo, pretende-se desenvolver o controle de adaptação dos intervalos de tempo para a tomada de decisão, estender os experimentos à outras máquinas quadri-processadas mais recentes, e avaliar o sistema com aplicações reais. Finalmente, a médio prazo, o protótipo seria avaliado em máquinas multiprocessadas com mais de quatro processadores.

Referências

[1] R. Berrendorf and B. Mohr. Pcl - the performance counter library: A common interface to access hardware performance counters on microprocessors (version 2.2). 2003.

[2] F. Cappello, O. Richard, and D. Etiemble. Investigating the performance of two programming models for clusters of smp pcs. In *Proc. of the 6th Int. Symposium on High Performance Computer Architecture Conference, Toulouse, France*, pages 349–359, January 2000.

[3] L. DeRose. The hardware performance monitor toolkit., August 2001.

[4] P. Dinda, T. Gross, R. Karrer, B. Lowekamp, N. Miller, P. Steenkiste, and D. Sutherland. The architecture of the remos system. In *Proc. 10th IEEE Symp. on High Performance Distributed Computing.*, 2001.

[5] M. K. Gardner, W. Feng, M. Broxton, A. Engelhart, and G. Hurwitz. Magnet: A tool for debugging, analysis and adaptation in computing systems. In *Proc. of the 3rd IEEE/ACM International International Symposium on Cluster Computing and the Grid (CCGrid 2003) - Tokyo Japan*, pages 12–15, May 2003.

[6] Intel Corporation. http://www.intel.com/, 2002.

[7] R. Jin and G. Agrawal. Performance prediction for random write reductions: a case study in modeling shared memory programs. In *Proc. of the 2002 ACM SIGMETRICS int. conf. on Measurement and modeling of computer systems, New York, USA*, pages 117–128, 2002.

[8] K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, and T. Spencer. End-user tools for application performance analysis, using hardware counters. International Conference on Parallel and Distributed Computing Systems, August 2001.

[9] B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. In *IEEE Computer.*, volume 28, pages 37–46, 1995.

[10] D. S. Nikolopoulos and C. D. Polychronopoulos. Adaptive scheduling under memory pressure on multiprogrammed clusters. In *Proc. of the Second IEEE/ACM Int. Symp. on Cluster Computing and the Grid (CCGrid 2002), Berlin, Germany, (Best Paper Award)*, May 2002.

[11] M. Pattersson. Linux x86 performance-monitoring counters driver. http://user.it.uu.se/~mikpe/linux/perfctr/.

[12] M. Pillon, O. Richard, and G. Da-Costa. Drac: Adaptive control system with hardware performance counters. In *International Conference on Parallel and Distributed Computing (EuroPar 2004), Pisa, Italy, 31st August - 3rd September*, Lecture Notes in Computer Science. Springer, 2004. to appear.

[13] R. L. Ribler, H. Simitci, and D. A. Reed. The autopilot performance-directed adaptive control system. In *Future Generation Computer Systems.*, volume 18, pages 175–187, 2001.

[14] W. Smith. A framework for control and observation in distributed environments. In *Technical Report NASA Advanced Supercomputing Division, NASA Ames Research Center NAS-01-006, July.*, 2001.

[15] O. Specifications. Simple, portable, scalable smp programming. http://www.openmp.org/.