Uso de Contadores de Hardware para Análise de Desempenho e Otimização

Lisandro L. Trarbach, Marcelo C. Zembrzuski, André L. Martinotto, Delcino Picinin, Tiarajú A. Diverio

Instituto de Informática - Universidade Federal do Rio Grande do Sul
CP 15064, 91501-970, Porto Alegre, RS, Brasil
lltrarbach@inf.ufrgs.br

Introdução

Contadores de *Hardware* estão disponíveis na maioria dos processadores modernos. Esses contadores fornecem informações que podem ser utilizados por desenvolvedores para uma análise detalhada de desempenho e, principalmente, na otimização de de programas [DON 2001]. Eles existem em processadores Intel desde a família 286. Uma das formas de acessar esses contadores é através do uso de bibliotecas. Para o desenvolvimento desse trabalho utilizou-se a biblioteca PAPI (*Performance Aplicattion Programming Interface*).

Como estudo de caso, optou-se por mostrar algumas informações obtidas através desses contadores em duas variantes do algoritmo de multiplicação de matrizes. Além disso é feito um estudo comparativo dessas com a operação de multiplicação de matrizes disponível na biblioteca BLAS (*Basic Linear Algebra System*). Com essas informações é possível compreender o motivo da diferença de desempenho das diferentes implementações dessa operação.

O presente artigo está estruturado da seguinte forma. Primeiramente é introduzida a biblioteca PAPI. Após são feitas algumas considerações sobre os algoritmos de multiplicação de matrizes utilizados. E por fim, são apresentados os resultados obtidos e conclusões.

PAPI

A biblioteca PAPI fornece uma API padrão e portável para acesso a contadores de *hardware*. Através dela, é possivel obter informações relativas a número de instruções, número de operações de ponto flutuante, movimentação de dados entre os níveis de memória cache, número de ciclos, entre outras. Para uma lista completa dos eventos disponíveis na biblioteca PAPI consulte [PAP 2003].

Multiplicação de Matrizes

Há muitos algoritmos para efetuar a operação de multiplicação de matrizes. Esses se diferenciam pelo número de operações executadas e pela forma como acessam os dados nos diferentes níveis de memória [ALV 2002].

O algoritmo convencional para calcular a multiplicação de matrizes consiste em três laços aninhados, todos iterando N vezes, sendo N a ordem das matrizes que compõem a operação. Existem seis variantes do algoritmo convencional. Elas se diferenciam apenas pela ordem na qual os laços são executados. Duas variantes bastante utilizadas são a *ijk* e *ikj*. Na variante *ijk* ao final de cada execução do laço mais interno é calculado um elemento da matriz resultante [ALV 2002]. Já a variante *ikj* se baseia no produto de um elemento da matriz A por um vetor da matriz B, gerando ao final de cada execução do laço mais interno um vetor parcial da matriz de resultado [CON 2003].

Três métodos são analisados neste artigo: os dois primeiros correspondem as variantes *ijk* e *ikj* e o terceiro faz uso da função *cblas-sgemm* da biblioteca BLAS [DON 2001].

Resultados Obtidos

Nessa seção são apresentados os resultados obtidos neste trabalho. No desenvolvimento do mesmo utilizou-se o a biblioteca PAPI 2.3.4.3 e a distribuição ATLAS 3.4.2 da biblioteca BLAS. Os algoritmos foram implementadas em linguagem C, utilizando o compilador *gcc* 2.95.4 sobre o sistema operacional Debian Linux com *kernel* 2.4.21. Os testes foram realizados utilizando um máquina Dual Pentium III 1.1GHz, com 1 GBytes de RAM e cache de 512KBytes. Essa máquina disponibiliza dois contadores de *hardware* de modo que dois eventos podem ser contados simultaneamente.

O gráfico 1 mostra os tempos de execução para as três implementações de multiplicação de matrizes para dimensões de matriz que variam desde 50 até 950. Em todos os casos analisados a implementação com BLAS obteve um resultado extremamente superior às duas variantes do algoritmo convencional. Além disso, observa-se que a variante *ikj* tem um desempenho consideravelmente superior ao da variante *ijk*.

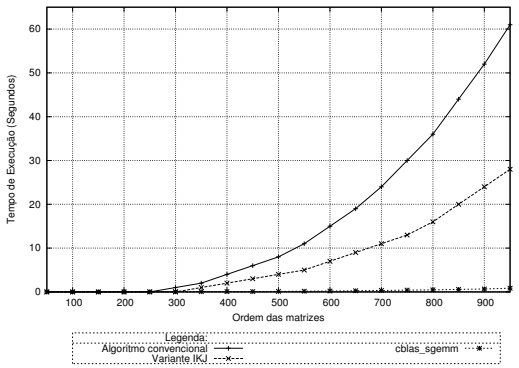

Figura 1: Tempo de Execucação

Analisando os gráficos da fig 2 e 3 podemos identificar a causa do melhor desempenho da variante *ikj* em relação a variante *ijk*. Esses gráficos representam o total de *cache misses* de nível 1 e 2, respectivamente. *Cache Misses* é uma solicitação de dados, feita pelo processador, que não estão presentes na cache. Observa-se que o número de *cache misses* de nível 1 é bem menor, para a variante *ikj*. Fato semelhante ocorre com o

número de *cache misses* de nível 2. Sabe-se que cada *cache miss* de nível 1 implica em um acesso à *cache* de nível 2, que é mais lenta que a de nível 1. Da mesma forma, cada *cache miss* no nível 2, implica em um acesso à memória principal, o que causa grande perda de desempenho.

Nas fig 4 e 5 podemos observar gráficos que comparam a *cache misses* de nível 1 e 2 da implementação da variante *ikj* com a implementação com uso da função *cblas-sgemm* da biblioteca BLAS. Como anteriormente podemos notar que a diferença no desempenho ocorre devido a grande diferença que existe entre o número de *cache misses* de níveis 1 e 2 das duas implementações. A implemetanção com a BLAS apresenta um número muito menor de *cache misses*.

O gráfico da fig 6 apresenta o processamento efetivo de cada uma das variantes. Entende-se por processamento efetivo a razão entre o número de ciclos que o processador não está ocioso e o total de ciclos de execução da aplicação. Observa-se no gráfico que a BLAS tem processamento efetivo quase total. Observa-se também, que a variante *ikj* tem um desempenho constante, mas bem abaixo do desempenho obtido com a BLAS. Como era de se esperar, a variante *ijk* teve o pior desempenho entre as três implementações testadas. Um fato interessante é que o número de ciclos aproveitados nesta variante cai enormemente para dimensões maiores que 200. Uma possível explicação para esse fato é que para tais dimensões, o tamanho da matriz é próximo ou maior que o tamanho da memória *cache*.

Em comparação feita entre essas três implementações com e sem uso do PAPI, observamos que o *overhead* causado pela utilização desta biblioteca é mínimo.

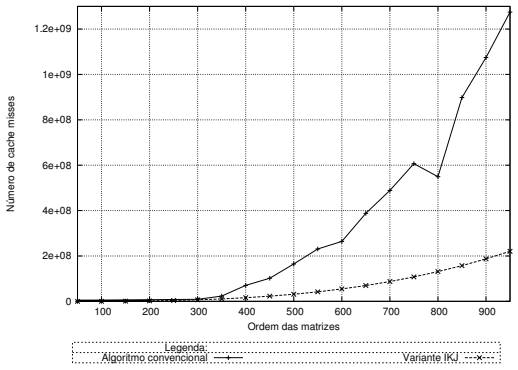

Figura 2: Cache Misses L1

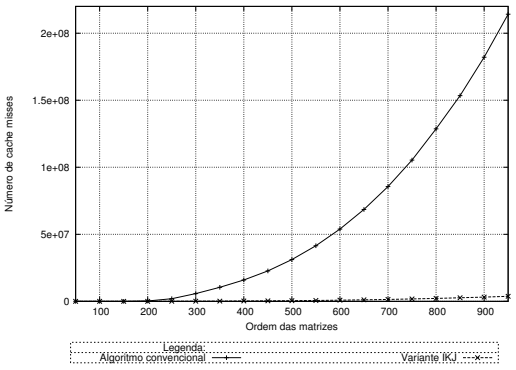

Figura 3: Cache Misses L2

Conclusões

O uso de contadores de *hardware* pode trazer informações úteis sobre a aplicação. Com o auxilio destes é possível monitorar o desempenho de trechos isolados de código. Através da análise dessas informações pode-se identificar os trechos em ocorre grande perda de desempenho. De posse de tais informações é possível otimizar esse código. Constatou-se que a biblioteca PAPI fornece uma forma fácil e rápida para obter acesso a esses contadores.

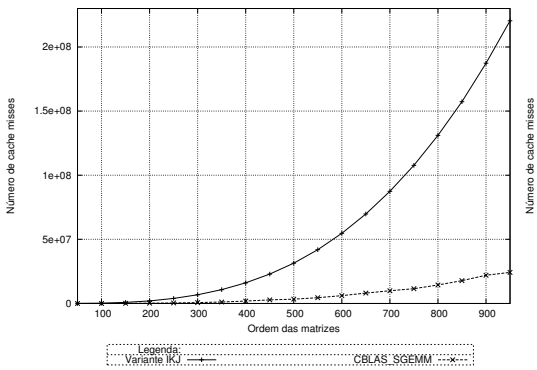

Figura 4: Cache Misses L1

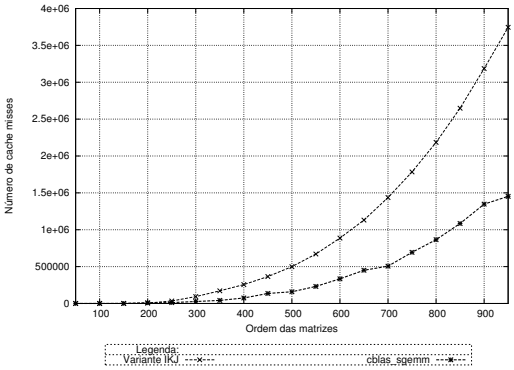

Figura 5: Cache Misses L2

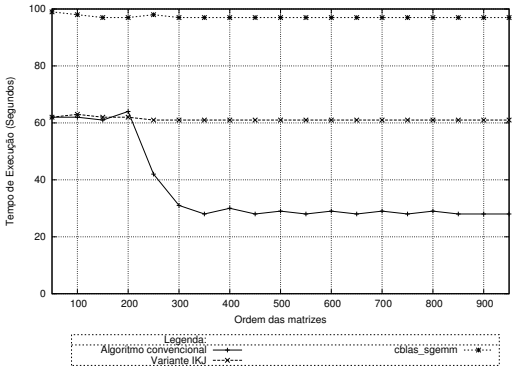

Figura 6: Uso efetivo da CPU

Referências

[ALV 2002] ALVES, R. et al. Multiplicação de matrizes: técnicas de otimização e paralelização de algoritmos. **Scientia**, São Leopoldo, v.13, n.1, p.17–32, 2002.

[CON 2003] CONTESSA, D. F. **Uso Eficiente da Biblioteca de Rotinas Básicas de Álgebra Linear no Ambiente de Programação Paralela por Troca de Mensagens em Cluster**. 2003. Trabalho de Conclusão (Curso de Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Porto Alegre, RS.

[DON 2001] DONGARRA, J. et al. A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters. In: LINUX CLUSTERS: THE HPC REVOLUTION, 2001. **Anais...** [S.l.: s.n.], 2001.

[PAP 2003] PAPI User's Guide. Disponível em: http://icl.cs.utk.edu/projects/papi/documents/. Acesso em: out. 2003.