Avaliação do Impacto da Afinidade de Processador no Desempenho de Aplicações Paralelas Executadas em Ambientes de Memória Compartilhada

Carlos Alexandre de Almeida Pires[1], **Marcelo Lobosco**[1]

[1]Grupo de Educação Tutorial do Curso de Engenharia Computacional
Universidade Federal de Juiz de Fora (UFJF) – Juiz de Fora, MG – Brasil

carlos.alexandre@engenharia.ufjf.br, marcelo.lobosco@ice.ufjf.br

Abstract. *Cache memories were introduced in order to reduce the high memory access costs. The scheduling algorithm used by the operating system (OS) can impact the performance of parallel applications in multicore processors, since cached data can be lost if the OS schedules the process to be executed in a distinct core. This paper evaluates the impact of affinity scheduling on the performance of parallel applications executiong on multicore processors. The results have shown that some applications can be very sensitive to the scheduling used. Gains up to 2.1 times were observed when affinity scheduling was used.*

Resumo. *Memórias cache foram adicionadas aos processadores com o objetivo de reduzir os altos custos de acesso à memória. A política de escalonamento efetuada pelo Sistema Operacional (SO) pode impactar o desempenho de aplicações paralelas em máquinas compostas por múltiplos núcleos, visto que o aproveitamento dos dados previamente mantidos em cache pode se perder caso o SO opte por escalonar o processo em outro núcleo. Este trabalho investiga o impacto da afinidade de processador no desempenho de aplicações paralelas executadas em tais ambientes. Os resultados mostram que as aplicações podem sofrer grandes impactos, com ganhos de desempenho de até 2.1 vezes sendo observados quando a política de afinidade é utilizada.*

1. Introdução

Entre os anos de 1986 e 2002, o desempenho dos uniprocessadores cresceu, em média, 52% ao ano, caindo desde então para uma taxa de 22% ao ano [Patterson and Hennessy 2005]. Barreiras impostas pela potência fizeram com que os fabricantes de processadores optassem por aumentar o número de processadores em um único *chip* como forma de possibilitar o contínuo aumento do desempenho dos processadores [Patterson and Hennessy 2005]. Problemas que até então eram proibitivos sob a ótica do desempenho tornaram-se viáveis com os ganhos observados nas últimas décadas. Mas do mesmo modo que o poder computacional vem aumentando, também crescem as demandas por maior poder de processamento.

Uma ferramenta que pode ser utilizada para reduzir os custos de execução de uma aplicação é a computação paralela. Uma das formas de reduzir o tempo de computação é dividir a computação a ser feita entre os múltiplos núcleos de processamento disponíveis nos processadores atuais. Em alguns cenários, uma aplicação paralela, por melhor que

seja o seu código, pode ter seu desempenho comprometido por conta das opções de escalonamento efetuadas pelo sistema operacional. Esse cenário pode ocorrer, por exemplo, quando o sistema operacional (SO) decide escalonar um processo ou *thread* da aplicação paralela, a cada fatia de tempo, em um núcleo diferente. O impacto no desempenho neste caso surge pelo fato de cada núcleo possuir uma *cache* privativa. Assim, ao ser escalonada em um núcleo diferente, a *thread* não tem mais acesso seus aos dados previamente mantidos em *cache*, devendo compulsoriamente pagar o preço de novos acessos à memória.

Uma das formas de resolver esse problema é empregar a técnica conhecida como afinidade de processador (*Processor Affinity*) [Chandra et al. 1994] no escalonamento de processos. Neste caso, o SO opta por escalonar a *thread* sempre no mesmo processador. Nesse trabalho, iremos avaliar o impacto da afinidade de processador no desempenho de um conjunto de aplicações paralelas.

2. Trabalhos Relacionados

Vários trabalhos anteriores avaliaram, sob diversos aspectos, o impacto da afinidade de processador no desempenho de aplicações paralelas. Um dos trabalhos avaliou o impacto de diversas políticas de escalonamento no desempenho de aplicações paralelas [Gupta et al. 1991], sendo uma delas a política baseada na afinidade de processador. Os resultados do artigo mostraram que a afinidade de processador foi responsável pelo melhoramento das taxas de acerto da *cache* e pelo aumento da utilização do processador. Como o estudo foi realizado no início da década de 1990, a arquitetura utilizada nos testes difere da utilizada neste estudo. Um estudo mais recente [Ribeiro et al. 2009] realizado em uma arquitetura ccNUMA apresenta o impacto da afinidade em um conjunto de aplicações científicas. Este artigo realiza o estudo em uma arquitetura UMA. Outro estudo mostrou que o uso da afinidade de processador pode impactar positivamente o desempenho das aplicações [Squillante and Lazowska 1993]. Esse artigo fez um estudo teórico, sem realizar uma análise voltada para a execução de aplicações paralelas, nem baseada em máquinas reais, como a feita neste trabalho.

3. Método

Para avaliar quantitativamente o impacto da afinidade de processador no desempenho de aplicações paralelas, foi realizado o seguinte teste. Foram medidos os tempos de execução de um conjunto de aplicações paralelas, primeiro sem o uso de afinidade de processador no escalonamento das *threads* pelo SO e depois com o uso da afinidade. Foram utilizados aplicativos do *benchmark* NPB (*NAS Parallel Benchmarks*) [Bailey et al. 1991, Jin et al. 1999] implementados em OpenMP [Dagum and Menon 1998] para os testes. O *benchmark* foi executado com 1, 2, 4 e 8 *threads*. Para habilitar o uso da afinidade de processador, foi utilizada a variável de ambiente do OpenMP **GOMP_CPU_AFFINITY**.

Os testes foram feitos em um nó de um *cluster*, cujo acesso exclusivo é feito através da submissão de *jobs* a uma fila de tarefas. O nó escolhido para execução possui dois processadores Intel Xeon E5620 de 2.40 GHz, cada um com quatro núcleos, totalizando assim 8 núcleos de processamento. Cada núcleo possui 32KB de *cache* L1 de dados, 32 KB de *cache* L1 de instruções, e 256KB de *cache* L2, esta compartilhada entre instruções e dados. Cada processador possui uma *cache* L3 de 12MB, usada para armazenar tanto dados como instruções, e compartilhada entre os quatro núcleos do processador. Apesar deste processador possuir suporte para a tecnologia *hyper-threading*

[Marr et al. 2002], esta foi desabilitada na BIOS. A máquina executa o SO Linux com *kernel* na versão $2.6.32$. O compilador **gFortran** na sua versão $4.4.7$ foi usado para compilar os programas. O tempo foi computado através do aplicativo **time**, disponível no SO. Para fins de cálculo do tempo de execução, foi considerada a média aritmética simples de 5 execuções, sendo o desvio-padrão observado foi menor que $1,9\%$.

3.1. NPB 3.3

Para avaliar o impacto da afinidade de escalonamento no desempenho das aplicações, foi utilizada a versão 3.3 do NPB, um conjunto de *benchmarks* criado pela Divisão de Supercomputação Avançada da NASA (*National Aeronautic and Space Administration*). Os *benchmarks* são compostos por *kernels* e pseudo-aplicações. *Kernels* são trechos de códigos que podem ser encontrados em diversas aplicações. Neste artigo, escolhemos três *kernels* e três pseudo-aplicações para os testes. Os *kernels* escolhidos foram EP, CG, e FT. As pseudo-aplicações utilizadas foram BT, SP e LU. Os *kernels* e pseudo-aplicações foram implementadas em Fortran e são descritas a seguir.

- *Embarrassingly Parallel* (EP): Gera pares de valores aleatórios seguindo uma distribuição gaussiana a partir do esquema polar de Marsaglia. Este *kernel* realiza poucas operações de comunicação ao longo de sua execução.
- *Conjugate Gradient* (CG): Implementa o método do gradiente conjugado para calcular uma aproximação para o menor autovalor em uma matriz esparsa, não estruturada e com valores gerados aleatoriamente. Possui acesso irregular à memória.
- *Fourier Transform* (FT): Implementa o algoritmo da transformada rápida de Fourier em três dimensões. O *kernel* resolve três transformadas separadamente, uma para cada dimensão. Os acessos à memória são bem distribuídos e regulares.
- *Block Tri-diagonal solver* (BT): Aplicação que implementa uma simulação computacional de dinâmica de fluidos (CFD - *Computational Fluid Dynamics*) usando para isso as equações de Navier-Stokes em três dimensões. O método das diferenças finitas com aproximações pelo método ADI (*Alternating Direction Implicit*) é empregado na solução.
- *Scalar Penta-diagonal solver* (SP): É uma aplicação semelhante a BT, porém usando um método de aproximação distinto, o método de Beam-Warming. A aplicação acessa a memória de modo regular.
- *Lower-Upper solver* (LU): Esta aplicação também implementa uma simulação computacional de dinâmica de fluidos. O método Sobre-relaxação Sucessiva (SSOR) é utilizado para resolver uma discretização por diferenças finitas das equações de Navier-Stokes em três dimensões, separando-as em duas matrizes: triangular superior e triangular inferior.

Todos os *benchmarks* possuem um parâmetro que define o tamanho das entradas a serem utilizadas na sua execução. Para os testes efetuados nesse trabalho, foi usado o tamanho B.

4. Resultados

A Tabela 1 apresenta os resultados iniciais. Os resultados apresentam, para cada *benchmark*, os tempos médios de computação, com e sem o uso de afinidade, para configurações com 1, 2, 4 e 8 *threads*. Pode-se observar que, na configuração com uma *thread*, os tempos obtidos com e sem o emprego da afinidade de processador podem ser considerados

iguais, tendo em vista que o desvio-padrão foi inferior à $1,9\%$. De modo geral, os tempos de execução para 2 e 4 *threads* com o uso de afinidade de processador foram muitas vezes superiores ao tempo de execução quando a afinidade não foi usada, comportamento este oposto ao que seria esperado. Apenas na configuração com 8 *threads* que o tempo de execução com o uso de afinidade foi inferior ao tempo de execução sem o seu emprego. Para esta configuração, os maiores ganhos observados foram obtidos nos *benchmarks* BT e EP: 2.1 e 1.3, respectivamente. Na execução de FT o ganho foi de aproximadamente 1%, ficando abaixo do desvio-padrão. Para os demais *benchmarks*, os ganhos variaram entre 2% e 3%, sendo ligeiramente superiores ao desvio-padrão.

Tabela 1. Tempos médios de execução, em segundos, para cada *benchmark*.

Benchmark	Uso de Afinidade	Número de *Threads*			
		1	2	4	8
BT	Não	307,1	157,4	81,6	95,6
	Sim	307,2	158,4	87,8	46,3
CG	Não	97,9	50,7	28,1	21,2
	Sim	97,9	52,5	36,7	20,6
EP	Não	87,4	44,3	22,3	14,9
	Sim	87,3	44,1	23,2	11,6
FT	Não	81,1	41,4	21,8	14,3
	Sim	81,0	42,7	27,0	13,8
LU	Não	291,4	149,9	77,4	45,1
	Sim	291,4	150,8	83,3	44,4
SP	Não	262,2	136,4	74,1	50,9
	Sim	262,2	140,9	91,4	49,6

A análise detalhada dos resultados, da arquitetura do processador utilizado nos testes e dos *scripts* utilizados na execução dos *benchmarks* levou à hipótese de que as execuções com 2 e 4 *threads* com o uso de afinidade estavam sendo executadas em apenas um dos processadores. Isto poderia ser uma desvantagem em comparação com a execução sem afinidade, caso o SO escalone as *threads* em processadores diferentes, tendo em vista que neste caso o dobro de dados poderia estar presente na *cache* L3.

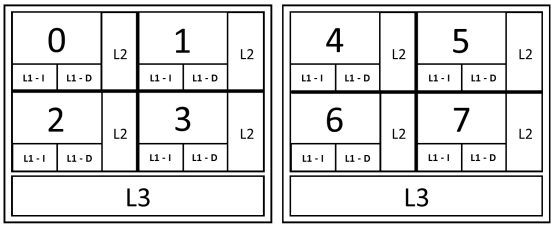

Figura 1. Arquitetura do processador E5620. Dois processadores são representados, totalizando 8 núcleos. São destacadas as *caches* L1 (dados e instruções), L2 e L3. As *caches* L1 e L2 são de acesso privativo a cada núcleo, enquanto a L3 é compartilhada por todos os núcleos do processador.

De fato, no *script* de submissão das tarefas foi utilizada a seguinte configuração: **export GOMP_CPU_AFFINITY=0-7**. Esta configuração equivale ao uso da estratégia *compact* para uso com a variável de ambiente **KMP_AFFINITY**, disponível apenas no

compilador Intel (este trabalho usou o compilador GNU). Nas configurações com 2 e 4 *threads*, isto implica que as *threads* executarão em um único processador, visto que os núcleos de número 0 à 3 estão presentes no mesmo processador, conforme apresentado na Figura 4. Para confirmar a validade da hipótese, um novo teste foi executado, mas desta vez alterando os valores usados na primitiva que habilita o uso da afinidade, de forma a fazer com que processadores distintos sejam utilizados nas configurações com 2 e 4 *threads*: **export GOMP_CPU_AFFINITY=0 4 1 5 2 6 3 7**. Esta configuração equivale ao uso da estratégia *scatter* para uso com a variável de ambiente **KMP_AFFINITY**. Os resultados obtidos para esta configuração são apresentados na Tabela 2.

Tabela 2. Tempos médios de execução, em segundos, para cada um dos *benchmarks*, executando com o segundo esquema de afinidade de processador.

Benchmark	Número de *Threads*			
	1	2	4	8
BT	307,2	156,9	80,1	46,3
CG	97,9	50,0	27,7	20,6
EP	87,3	43,9	21,9	11,6
FT	81,0	41,4	21,5	13,8
LU	291,4	148,8	76,7	44,4
SP	262,2	134,5	73,6	49,6

Pode-se observar, quando se comparam os resultados das Tabelas 1 e 2, que os tempos de execução para as configurações que fazem uso de afinidade e empregam 2 e 4 *threads* na execução melhoraram, o que confirmou a hipótese. Apesar da grande melhora verificada nestas configurações, os tempos de execução com afinidade podem ser considerados iguais aos tempos sem afinidade, visto que os ganhos foram inferiores ao desvio-padrão. Já os tempos de execução para as configurações com 1 e 8 *threads* se mantiveram iguais, conforme seria esperado.

Apesar dos resultados significativamente melhores nas execução de BT e EP com uso de afinidade de processador na configuração com 8 *threads*, as demais aplicações/*kernels* e configurações apresentaram tempos de execução muito próximos ou ligeiramente melhores que os tempos obtidos sem o uso de afinidade.

5. Conclusão

Este trabalho apresentou uma análise do impacto do escalonamento baseado em afinidade de processador no desempenho de um conjunto de aplicações paralelas executadas em um ambiente de memória compartilhada distribuída. A análise mostrou que, das seis aplicações e *kernels* avaliados, duas tiveram grandes ganhos de desempenho (2.1 e 1.3 vezes) quando executadas com a afinidade de processador habilitada em uma configuração com 8 *threads*, três tiveram pequenos ganhos, entre 2% e 3%, e em uma não houve ganho. Não foram observados ganhos de desempenho com as configurações com 1, 2 e 4 *threads*. O trabalho mostrou ainda que a simples habilitação da afinidade de processador não é suficiente para permitir ganhos de desempenho em ambientes compostos por múltiplos processadores, sendo necessária uma definição criteriosa da ordem em que os núcleos destes processadores serão utilizados. Possíveis causas para não ter ocorrido ganhos de desempenho para estas configurações são: a) o tamanho do conjunto de dados utilizado,

e b) as taxas de falhas na *cache*. Por exemplo, se uma parte significativa do conjunto de dados utilizados pela aplicação cabe completamente na *cache* L3, o uso da afinidade de processador pode perder um pouco do seu sentido, visto que esta *cache* é compartilhada entre todos os núcleos do processador. Outra situação em que não haveria vantagem no uso da afinidade de processador é quando as taxas de falha de *cache* são altas. Neste cenário, pode não fazer tanta diferença usar sempre os mesmos núcleos para executar as *threads*, visto que a memória principal sempre deverá ser acessada para recuperar os dados necessários para a execução da aplicação. Tais tópicos serão objeto de investigação em trabalhos futuros. Também será investigado o impacto do uso de afinidade de processador em arquiteturas distintas, como na família de processadores AMD *Bulldozer*. Nesta arquitetura, dois núcleos de processamento compartilham uma única unidade de ponto-flutuante. O uso da afinidade de processador tende a ser essencial nesta arquitetura, em especial para aplicações que a) fazem intensivo uso de ponto-flutuante, e b) que empreguem em sua execução uma quantidade de *threads* igual à metade do número de núcleos disponíveis. Se tais núcleos escolhidos pelo SO não tiverem uso exclusivo da unidade de ponto-flutuante, ocorrerá concorrência por seu uso, diminuindo assim o desempenho da aplicação.

Referências

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., et al. (1991). The nas parallel benchmarks. *International Journal of High Performance Computing Applications*, 5(3):63–73.

Chandra, R., Devine, S., Verghese, B., Gupta, A., and Rosenblum, M. (1994). Scheduling and page migration for multiprocessor compute servers. In *ACM SIGPLAN Notices*, volume 29, pages 12–24. ACM.

Dagum, L. and Menon, R. (1998). Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55.

Gupta, A., Tucker, A., and Urushibara, S. (1991). The impact of operating system scheduling policies and synchronization methods of performance of parallel applications. *SIGMETRICS Perform. Eval. Rev.*, 19(1):120–132.

Jin, H.-Q., Frumkin, M., and Yan, J. (1999). The openmp implementation of nas parallel benchmarks and its performance.

Marr, D. T., Binns, F., Hill, D. L., Hinton, G., Koufaty, D. A., Miller, J. A., and Upton, M. (2002). Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1).

Patterson, D. and Hennessy, J. (2005). *Organização e projeto de computadores: a interface hardware/ software*. Ed. Campus, RJ.

Ribeiro, C. P., Mehaut, J.-F., Carissimi, A., Castro, M., and Fernandes, L. G. (2009). Memory affinity for hierarchical shared memory multiprocessors. *Computer Architecture and High Performance Computing, Symposium on*, 0:59–66.

Squillante, M. S. and Lazowska, E. D. (1993). Using processor-cache affinity information in shared-memory multiprocessor scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, 4(2):131–143.