Avaliação do Escalonador de Processos do Sistema Operacional Linux

Henrique Weber, Marco Antonio Zanata Alves, Philippe Olivier Alexandre Navaux

Apresentação

A atual demanda por performance, liderada por aplicações que tornam-se cada vez mais sofisticadas e complexas, requer técnicas de programação e recursos extremamente eficientes e concisos Nesta área, a indústria está apostando no desenvolvimento de arquiteturas *multi-core* com emissão de múltiplas instruções, escalonamento dinâmico, execução especulativa e *caches* não-bloqueantes.

Muitas aplicações *multi-core* são projetadas visando uma divisão de trabalho que alcance a melhor performance durante a execução do programa. Neste contexto, o programador pode definir como separar os dados, com o objetivo de permitir a execução simultânea e o menor número de dependências possível. Porém, esta divisão nem sempre ocorre de forma ótima, uma vez que os dados são utilizados mais de uma vez durante a execução do programa.

Portanto, caso o programador decida deixar esta tarefa a cargo do sistema operacional, é importante usar o conhecimento sobre a arquitetura com o objetivo de fazer esse mapeamento de forma manual, uma vez que um uso mais eficiente da *cache* L2 permite um ganho de desempenho. Deste modo, o objetivo deste trabalho é analisar como o sistema operacional *Linux* distribui os processos entre os núcleos disponíveis na máquina.

Ambiente de execução

Foram feitos testes em uma máquina com dois processadores *Intel Quad-core Xeon E5405* (*Harpertown*), frequência de 2GHz e 8GB de memória principal. Cada par de núcleos compartilha 6MB de *cache* L2. O sistema operacional utilizado foi *Linux*, *kernel* versão *2.6.31-22-generic*, distribuição Ubuntu.

O *benchmark* utilizado foi o Modelo Geral de Circulação Atmosférica (AGCM), também conhecido como Modelo Global.

Metodologia

Executar o *benchmark* para combinações de mapeamento com 1,2,4 e 8 núcleos. Os resultados finais são baseados na média destas 10 execuções, e são divididos em quatro diferentes distribuições de núcleos utilizados nos testes, que são os seguintes:

Balanced: Os núcleos a serem usados não compartilham *cache* L2 e não estão localizados no mesmo processador;

Semi-balanced: Os núcleos não compartilham *cache* L2 mas estão no mesmo processador;

Unbalanced: Os núcleos compartilham *cache* L2 e estão localizados no mesmo processador;

Chosen by the OS: Os núcleos a serem usados são escolhidos pelo sistema operacional.

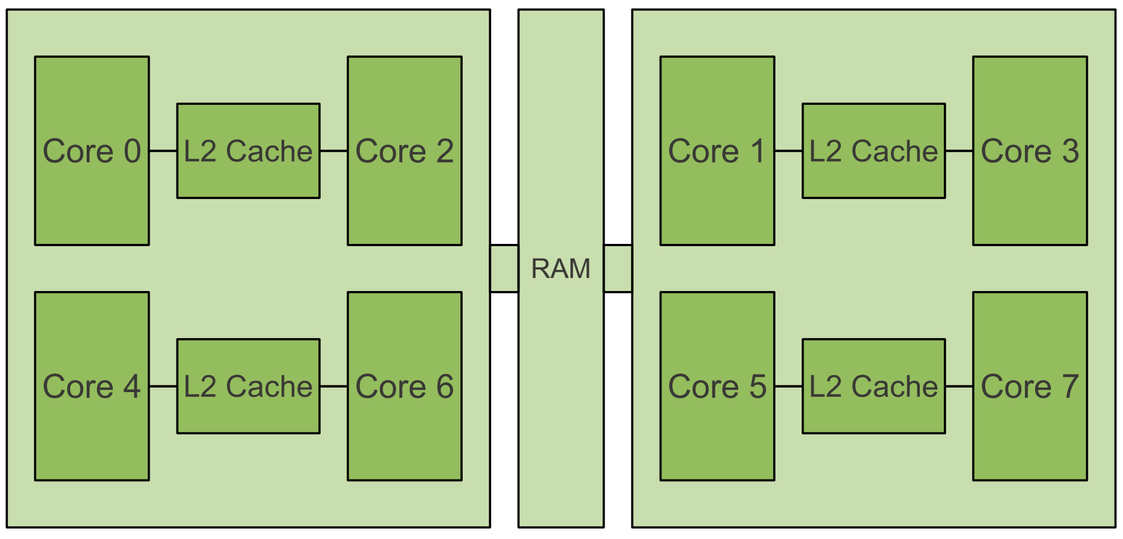

Figura 1: Topologia da máquina de teste

Resultados

Para a aplicação executando em 2 núcleos, o *speedup* foi próximo de 2 como esperado. Com 4 núcleos, o *speedup* não cresce linearmente, chegando em torno de 3,5. Para 8 núcleos, o *speedup* alcança apenas 5. Este resultado aquém do esperado pode estar relacionado com o conflito por recursos: 2 ou mais processos compartilhando a mesma *cache* implica grande número de *cache misses*; Baixo número de controladores de memória pode ser um gargalo no fluxo de dados.

Entre todos os mapeamentos avaliados, a melhor performance foi alcançada com o mapeamento "*Balanced*", sendo seguido pelo mapeamento "*Chosen by the OS*". A pequena diferença entre "*Balanced*" e "*Chosen by the OS*" deve-se ao fato de que nem sempre o sistema operacional faz o mesmo balanceamento . Desta forma, percebe-se que a afinidade para os processos reduz esta variação entre as execuções.

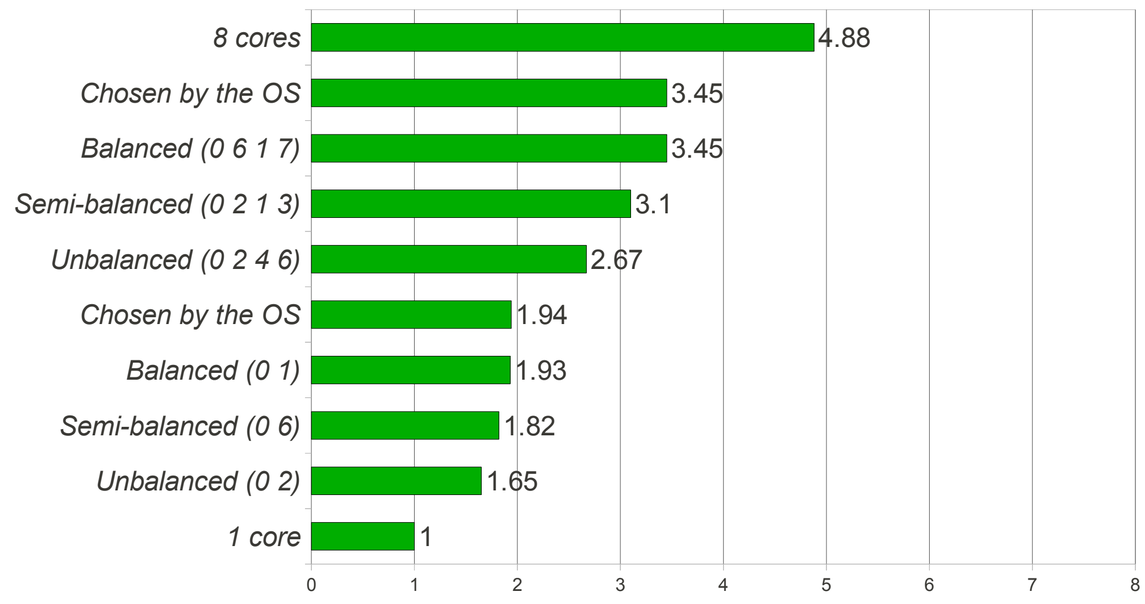

Figura 2: Tabela de speedup para testes utilizando 1, 2, 4 e 8 núcleos.

Conclusões

Sabendo que as máquinas *multi-core* são uma das soluções para prover um aumento contínuo na performance dos processadores atuais e futuros, estudos sobre um uso mais eficiente desta tecnologia, como apresentado, são muito importantes.

Os testes conduzidos neste trabalho trouxeram resultados de performance do escalonamento manual e o realizado pelo sistema operacional. Desta forma, pode ser mostrado que o sistema operacional *Linux* possui um escalonador que mapeia os processos tentando isolá-los tanto quanto for possível, fazendo com que cada processo possa utilizar uma *cache* L2 exclusiva, obtendo, em muitos casos, uma boa performance.

Referências:

S. Boyd-Wickizer, R. Morris, and M. Kaashoek. Reinventing scheduling for multicore systems. In Proceedings of the 12th Workshop on Hot Topics in Operating Systems, Monte Verita, Switzerland, 2009.

J. Panetta, S. Barros, J. Bonatti, S. Tomita, and P. Kubota. Computational cost of CPTEC AGCM. In Use Of High Performance Computing In Meteorology: Proceedings of the Twelfth ECMWF Workshop, page 65. World Scientific Pub. Co. Inc, 2007.

K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. ACM SIGPLAN Notices, 31(9):11, 1996.

R. Souto, R. Avila, P. Navaux, M. Py, T. Diverio, H. Velho, S. Stephany, A. Preto, J. Panetta, E. Rodrigues, et al. Processing mesoscale climatology in a grid environment. 2007. R. Thekkath and S. Eggers. Impact of sharing-based thread placement on multithreaded architectures. In Proceedings of the 21st annual international symposium on Computer architecture, pages 176–186. IEEE Computer Society Press, 1994.