Utilização de afinidade no escalonamento de threads em multiprocessadores

Alan Schlindvein de Araujo, Gerson Geraldo H. Cavalheiro

Departamento de Informática – Universidade Federal de Pelotas
Caixa Postal 354 – 99010-900 – Pelotas – RS – Brasil

alansaraujo.schlindvein@anahy.org, gerson.cavalheiro@ufpel.edu.br

Resumo. *A multiprogramação leve é o modelo de programação mais adequado para utilização em arquiteturas multiprocessadas. Este artigo apresenta uma discussão sobre o uso de associação de afinidades entre fluxos de execução e processadores para otimizar as decisões de escalonamento. Os resultados desta pesquisa deverão ser incorporados ao ambiente de execução de Athreads.*

1. Introdução

Arquiteturas multiprocessadas (De Rose, 2002) são caracterizadas por possuir múltiplas unidades de processamento compartilhando acesso a um espaço de endereçamento compartilhado. Arquiteturas UMA (Uniform Memory Access) e NUMA (Non-Uniform Memory Access) consistem em organizações desta classe de arquitetura, onde as arquiteturas UMA representam organizações onde o tempo de acesso a qualquer posição de memória é uniforme e as arquiteturas NUMA organizações nas quais o espaço de endereçamento encontra-se distribuído em módulos de memória não eqüidistante de todos os processadores. Nos multiprocessadores, cada processador é independente para efetuar sua própria computação, lendo e escrevendo dados em todo espaço de endereçamento. Mais ainda, o próprio espaço de endereçamento serve como substrato às comunicações (troca de dados) entre processadores. Os populares processadores multicore também podem ser considerados multiprocessadores, uma vez que possuem múltiplas unidades completas de processamento compartilhando acesso a um espaço de endereçamento comum.

É comum o emprego de arquiteturas multiprocessadas no processamento paralelo e de alto desempenho. Para este fim, a multiprogramação leve é o modelo de programação que mais adaptado, uma vez que prevê a execução concorrente de fluxos de execução trocando dados via uma área memória global (Ungerer, 2003). Existem diversas ferramentas de programação que oferecem este modelo de programação, entre elas POSIX Threads (Nichols, 1998), ou Pthreads, é uma das mais populares. Esta ferramenta permite decompor a execução de um programa em termos de fluxos de execução concorrentes, usualmente denominados threads, submetidos à execução sobre os recursos de hardware disponíveis. Uma política de escalonamento é aplicada sobre estes threads para determinar o acesso dos threads aos processadores disponíveis.

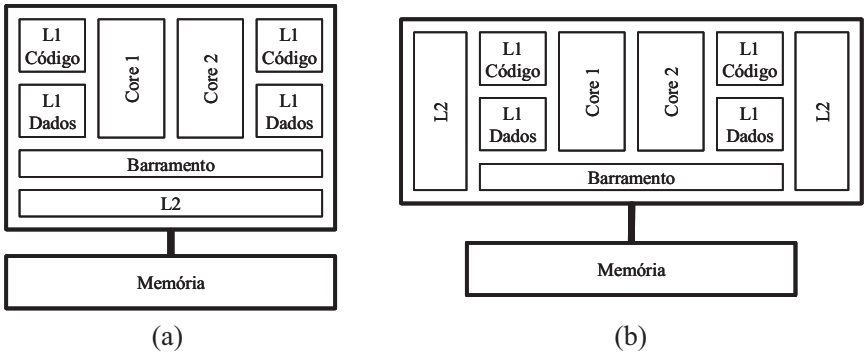
Este trabalho tem por finalidade identificar se a associação de afinidade de threads a processadores pode trazer benefícios, em termos de melhora de desempenho,

na execução de programas multithread. O estudo inicial está sendo realizado sobre processadores multicore e terá seqüência sobre arquiteturas NUMA.

2. Processadores Multicore

O mercado denomina "processadores multicore" os circuitos integrados que possuem dois ou mais núcleos de processamento completos no seu interior. Como uma arquitetura multiprocessada UMA ordinária, todos os núcleos de processamento, ou *cores*, compartilham acesso ao mesmo espaço de endereçamento.

Existem diversas opções de configuração de processadores multicore, variando a o número e a velocidade dos *cores*, mas, em particular neste trabalho, atenta-se a variedade de opções para organização das memórias cache. Além de diversas alternativas em termos de capacidade (tamanho em bytes) destas memórias, as opções atuais contemplam o uso de um (1) a três níveis de cache (L1, L2 e L3), havendo a possibilidade de compartilhamento de determinados níveis entre os *cores* ou de uso exclusivo. A Figura 1 ilustra o esquema de blocos de duas arquiteturas dualcore, onde uma possui níveis L1 e L2 dedicados e a segunda L2 compartilhado.



(a) (b)

Figure 1. Esquema de cache de processadores multicore com (a) e sem (b) compartilhamento de cache L2

Dado ao baixo custo e o potencial de paralelismo, este tipo de arquitetura vem sendo incorporado a soluções para o processamento de alto desempenho. No entanto, deve-se atentar que um limitador do poder de processamento dos multiprocessadores está associado à capacidade da memória atender as múltiplas solicitações de acesso. Moore (2008) discute que a adequação deste tipo de processador para processamento de aplicações que manipulam intensivamente dados não é direta, em função dos atrasos de computação decorrentes de acesso de um grande número, oito ou mais cores, segundo este autor, à memória compartilhada.

Uma alternativa é explorar a localidade de acesso a dados pelos threads via estratégias de escalonamento. Desta forma, identificando threads a *cores* pode ser possível reduzir o número de *caches-misses* efetuados devido à movimentação de threads entre *cores*.

3. Threads e o Padrão Pthreads

Threads no padrão Pthreads consistem em unidades de descrição de concorrência de um programa. Quando em execução, os recursos de processamento de um determinado

processo são compartilhados entre todos os threads que o compõem. No entanto, cada thread é uma unidade de escalonamento independente, podendo, os threads, serem escalonados de três maneiras distintas, conforme o modelo de implementação empregado na ferramenta. Os modelos para implementação de threads são conhecidos como: 1:1, N:1 e N:M (Cavalheiro, 2001). No modelo 1:1, ou *one-to-one*, threads são denominadas threads kernel, manipuladas de maneira individual pelo sistema operacional, se um thread é bloqueado para a realização de outra operação as demais threads continuam suas atividades sem prejuízo. No modelo N:1, ou *many-to-one*, threads usuário são escalonadas no escopo do processo, quando ocorre o escalonamento deste. Ou seja, o sistema operacional reconhece apenas o processo como unidade de escalonamento, os threads consistem em unidades de escalonamento internas ao processo. Nesta situação, quando um thread é bloqueado em uma sincronização externa ao processo, todos os threads do processo bloqueiam igualmente. No modelo N:M, ou *many-to-many*, existe uma distinção uma dissociação entre a concorrência da aplicação, descrita por N threads usuário e o paralelismo de execução, caracterizadas por M threads sistema. Neste modelo, o sistema operacional é responsável pelo escalonamento dos threads sistema enquanto que no nível aplicativo é realizado o escalonamento dos threads usuário sobre os threads sistema.

Algumas primitivas definidas pelo padrão Pthreads são apresentados na seqüência. Respectivamente para criação, término e sincronização entre duas threads.

```
pthread_create( pthread_t *thread,const pthread_attr_t *atrib,
                void*(*funcao)(void*),void* args);
void pthread_exit(void *retval);
int pthread_join(pthread_t th, void **thread_return);
```

Como threads compartilham uma mesma área na memória, casos dois threads acessem o mesmo valor da memória, os trechos de código que realizam estes acessos são considerados sessões críticas. Cada thread deve obter exclusividade ao dado na memória, essa exclusividade é garantida pelo uso de **mutex**. As primitivas na seqüência permitem inicializar um mutex, obter e liberar um mutex, controlando o avanço da execução de threads sobre as sessões críticas.

```
pthread_mutex_init(pthread_mutex_t *m,pthread_mutexattr_t *atrib);
int pthread_mutex_lock(pthread_mutex_t *m);
int pthread_mutex_unlock(pthread_mutex_t *m);
```

Observa-se que o padrão define que, quando um thread é criado, a ele pode ser associado um atributo de escalonamento e de escopo. Este último atributo permite indicar se o thread deve ser escalonado como thread usuário ou thread sistema (modelos N:1 ou 1:1). O atributo de escalonamento permite indicar se o thread deve ser escalonado com a política FIFO, Round-Robin ou a provida pelo sistema operacional. O suporte a estes atributos é dependente da implementação da ferramenta Pthreads utilizada, devendo ser consultado as alternativas oferecidas por esta.

4. Afinidade ao Processador

A associação de afinidade de thread a um, ou a um grupo de, processador é possível na maioria dos sistemas operacionais disponíveis (Akhter, 2006). Em ambiente GNU-Linux as operações para manipular afinidade são:

```
int sched_setaffinity( pid_t pid, unsigned int tamanho, unsigned long *mascara );
int sched_getaffinity( pid_t pid, unsigned int tamanho, unsigned long *mascara );
```

Estas primitivas permitem, respectivamente, associar afinidade de um thread a um grupo de processadores e obter qual afinidade um thread possui. Os parâmetros

representam o identificador do processo, o tamanho, em bytes, da máscara de afinidade e a própria máscara. Esta máscara deve ser vista como uma seqüência de bits, cada bit correspondendo a um determinado processador/core, onde o bit menos significativo representa o processador/core um (1). Caso nesta máscara o valor de um determinado bit seja um (1), então o processador/core correspondente deve ser associado ao thread. Por exemplo, o valor `00000101` indica que o thread deve ser associado aos processadores/cores 1 e 3.

A expectativa é que o uso da afinidade permita explorar a localidade de referência a dados em memória e em cache resultando em um maior desempenho de execução. No entanto, alguns experimentos, como os realizados por Foong (2008) e Costa (2008) mostram que o ganho de desempenho não é significativo, havendo, ainda, perda de portabilidade do código. Foong (2008) ainda afirma que aplicar unicamente a técnica de controle de afinidade não garante melhora de desempenho de execução.

5. Conclusão

O trabalho de pesquisa documentado neste artigo esta sendo realizado no contexto do projeto Anahy e vislumbra-se integração dos resultados no núcleo de execução de Athreads (Cavalheiro, 2006). A continuidade será no estudo dos recursos que permitem, nos modernos processadores multicore, selecionar diferentes freqüências de trabalho para cada core. O objetivo a ser atingido está associado à obtenção de uma estratégia de escalonamento que considere a carga de trabalho gerada por um programa desenvolvido em Athreads para adequar a velocidade de operação de cada core de um chip multicore.

Referências

Akhter, S., Roberts, J. (2006). “Multi-core programming”. [s.l.]:Intel Press.

Cavalheiro, G. G. H. (2001). “Introdução à programação paralela e distribuída”. In. ERAD 2001. Porto Alegre:SBC.

Cavalheiro, G. G. H., Gaspary, L. P., Cardozo, M. A., Cordeiro, O. C. (2007) Anahy: a programming environment for cluster computing. In. High Performance Computing for Computational Science – VECPAR 2006. Berlin: Springer.

Costa, A. G. da. (2008) ”Implementação eficiente do modelo de Potts Celular em processadores multi-core”. Trabalho de Conclusão de Curso. BCC-UFPel.

De Rose, C. A. F. “Fundamentos de Processamento de Alto Desempenho”. In. ERAD 2002. Porto Alegre:SBC. 2002.

Foong, A., Fung, J., Newell, D. (2008) “Improved Linux* SMP scaling: User-directed processor affinity”. http://softwarecommunity.intel.com/articles/eng/1781.htm (acesso em 10/12/2008).

Moore, S. K. (2008) “Multicore Is Bad News For Supercomputers”. IEEE Spectrum Online. November.

Nichols, B., et. al. (1998) “Pthreads Programming”. Beijing: O'Reilly & Associates.

Ungerer, T., Robič, B., and Šilc, J. (2003) “A survey of processors with explicit multithreading”. ACM Comput. Surv. V.35(1):29-63.