Paralelização do problema inverso aplicado ao problema de absorção da água em processadores Multi-core

Ricardo Luis dos Santos[1], Paulo S. Sausen[1], Pedro A.P. Borges[2], Edson L. Padoin[1]

UNIJUÍ – Universidade Regional do Noroeste do Estado do Rio Grande do Sul
[1]DeTec – Departamento de Tecnologia – Ijuí, RS
[2]DeFEM – Departamento de Física, Estatística e Matemática – Ijuí, RS

ricardosan@brturbo.com.br, {sausen, pborges, padoin}@unijui.edu.br

Resumo. *Este artigo apresenta uma nova proposta de paralelização do problema inverso utilizado no cálculo da absorção da água pelas raízes em um sistema solo-planta, com base em dados do monitoramento do teor de água do solo. Tal problema já foi paralelizado utilzando o MPI como interface de troca de mensgens. Nesta nova versão, pretende-se paralelizar o algoritmo utilizando a programação multi-thread no intuito de explorar os recursos oferecidos pelos novos processadores multi-core.*

1. Introdução

Os resultados de pesquisas dos grandes fabricantes de processadores possibilitam que os consumidores tenham a disposição novos processadores com maiores capacidades de processamento. Dentre as inovações encontradas nos núcleos da arquitetura *Intel Core* da *Intel* pode-se destacar o *cache L2* compartilhado (4 *MB* e 2 *MB*), *pipeline* de 14 estágios, fusão de instruções (*macro-fusion*), *Enhanced SpeedStep, Advanced Digital Media Boost, Execute Disable Bit, Intel TXT* (*Trusted Execution Technology*), chaveamento elétrico avançado e desambiguação de memória [Torres, 2006]. Baseados nesta arquitetura foram lançados alguns núcleos entre eles o *Conroe, Allendale, Kentsfield e Merom* sendo este último a versão desta arquitetura para notebooks.

Já a AMD, lançou a *AMD64* que inclui entre outras tecnologias o processamento nativo em 32 e 64 bits, o controlador de memória integrado ao processador, a tecnologia *HyperTransport*, e as tecnologia *Cool'n'quiet*, *PowerNow!* e *Enhanced Virus Protection*. Verdadeiro, *dual-core*, seus dois *cores* são fabricados no mesmo *die* e não em separados, os dois núcleos são construídos sobre o mesmo *waffer* [Lima, 2006]. Com essa arquitetura foram lançados alguns núcleos entre eles *Manchester, Toledo, Windsor, Brisbane, Taylor, Trinidad* e *Tyler.*

Há algum tempo muitos programadores vem utilizando o conceito de múltiplas *threads* para obter um melhor desempenho em seus aplicativos. Essa tecnologia aliada a *multi-core*, faz com que os aplicativos sejam executados com um melhor desempenho fazendo com que as *threads* sejam executadas em paralelo, uma em cada núcleo do processador.

Na seção 2 são abordados os dois últimos escalonadores lançados utilizados no GNU/Linux, a sua definição e características. Na seção 3 é definido o problema. Na seção 4 é abordado sobre a implementação do algoritmo. Na seção 5 possui alguns estudos de casos com seus respectivos resultados, análises e um comparativo com os resultados obtidos e na seção 6 algumas considerações finais.

2. Escalonamento

O escalonador do SO Linux *kernel* 2.6, tornado público no final de 2003, foi chamado de O(1), pois mantém o tempo de seleção de um processo para executar constante, independentemente do número de processos. Este escalonador é baseado em tempo compartilhado, onde o tempo do processador é dividido em fatias de tempo (*quantum*) que são alocadas aos processos [Meira, 2008].

A partir da versão 2.6.23 do *kernel* do Linux, o escalonador O(1) foi substituído pelo *Completely Fair Scheduler* (*CFS*). De acordo com Ingo Molnar, "80% do projeto do *CFS* pode ser resumido em uma única frase: *CFS* basicamente modela em um hardware real um processador ideal e precisamente multi-tarefa" [Molnar, 2007].

Um processador ideal executa vários processos em paralelo, em que cada processo ocupa exatamente a mesma fração de energia do processador, ou seja, se apenas um processo está sendo executado, ele ocupa 100% do processador, mas se três processos estão sendo executados, cada um ocupa 33,3% do processador. Mas na prática, apenas uma tarefa pode ser executada de cada vez, o que torna essa situação injusta para os processos que devem ficar esperando. O *CFS* torna o escalonamento justo, quando um processo espera pela *CPU*, é calculado o tempo que ele ocuparia em um processador ideal (tempo calculado é igual ao tempo de espera dividido pelo número de processos esperando pelo processador, em nanossegundos), este é o tempo de execução destinado ao processo [Deives, 2007].

3. Definição do Problema

O problema definido para utilizar os recursos da programação multi-thread no intuito de explorar os recursos oferecidos pelos novos processadores multi-core foi o cálculo da absorção da água pelas raízes. Calculo este realizado através da paralelização do problema inverso– método de procura em rede, que demanda de grande poder de processamento [Borges, 2006].

O modelo matemático que descreve o movimento da água no solo, a paralelização do cálculo da retirada de água do solo pelas raízes de plantas, com base em dados do teor de água foi publicado em [Padoin, 2008].

O problema proposto neste trabalho é a paralelização de um algoritmo seqüencial, cuja finalidade é estimar a quantidade de água retirada do solo pelas raízes de plantas, com base em dados do monitoramento do teor de água do solo [Borges 2006].

4. Implementação Paralela

Os dados experimentais e resultados da implementação seqüencial demandam elevados tempos de processamento mesmo em uma malha com poucos pontos. Assim, torna-se difícil a exploração de resultados mais precisos com malhas maiores na resolução do problema direto, devido ao crescimento exponencial do tempo de execução e da grande dependência de dados imposta pela aplicação. Então, uma nova implementação paralela foi desenvolvida em linguagem C utilizando *threads*, onde o escalonamento das atividades foi efetivado de modo que cada *thread* receba os argumentos do processo principal e efetue o cálculo do problema direto retornando os resultados. Assim o processo principal recebe as soluções parciais e seleciona os melhores aproximando o resultado da solução ótima.

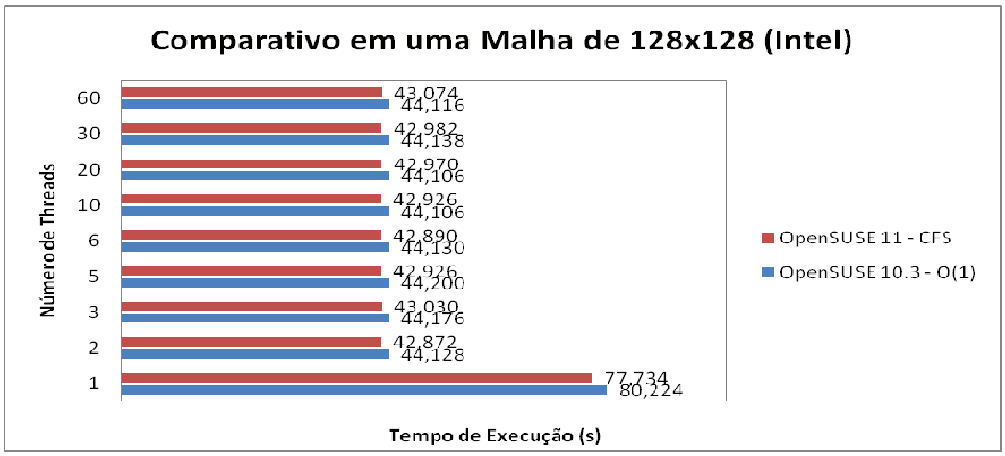
Primeiramente para programar o aplicativo com a utilização de *threads* foi utilizado a biblioteca *pthread*, do padrão POSIX, esta foi incluída no programa pelo comando: "#include <pthread.h>". Esta biblioteca inclui diversos comandos que são utilizados para gerenciar as *threads* entre eles os mais importantes são:

- "pthread_create(thread, attr, rotina, arg)": Permite criar uma *thread*, partindo do código de uma função existente no corpo do programa.
- "pthread_join(thread, retorno)": Faz com que o programa aguarde o término da *thread* indicada para dar seqüência para a execução.
- "pthread_t var": Cria uma variável do tipo pthread_t que é o descritor das *threads*.
- "pthread_attr_t var": Cria uma variável do tipo pthread_attr_t onde serão armazenadas as informações ou atributos utilizadas na criação das *threads*.

5. Resultados Obtidos

O estudo de caso a seguir foi realizado em um *notebook* com a seguinte configuração processador *Intel Mobile Core 2 Duo T7500* de 2.2 *GHz*, núcleo *Merom*, 4*MB* de *Cache L2*, *FSB* 800 *MHz*, 4*GB DDR2* de memória *RAM*, *OpenSUSE* 11 64 *bits* (*kernel* 2.6.25.5-1.1-*default* e gcc 4.3.1), 4*GB* de *Swap*. *OpenSUSE* 10.3 64 *bits* (*kernel* 2.6.22.5-31-*default* e gcc 4.2.1). Como se pode observar os resultados demonstram que utilizando threads o tempo de execução é consideravelmente reduzido em comparação a um programa seqüencial, 44,98% de redução no O(1) e 44,74% de redução no *CFS*. Na Figura 1 pode-se observar que em relação ao antigo escalonador, o O(1), o escalonador *CFS* possui um ganho de desempenho na casa de 3%, isso sem alterar nada de *hardware*, somente alterando o algoritmo de escalonamento.

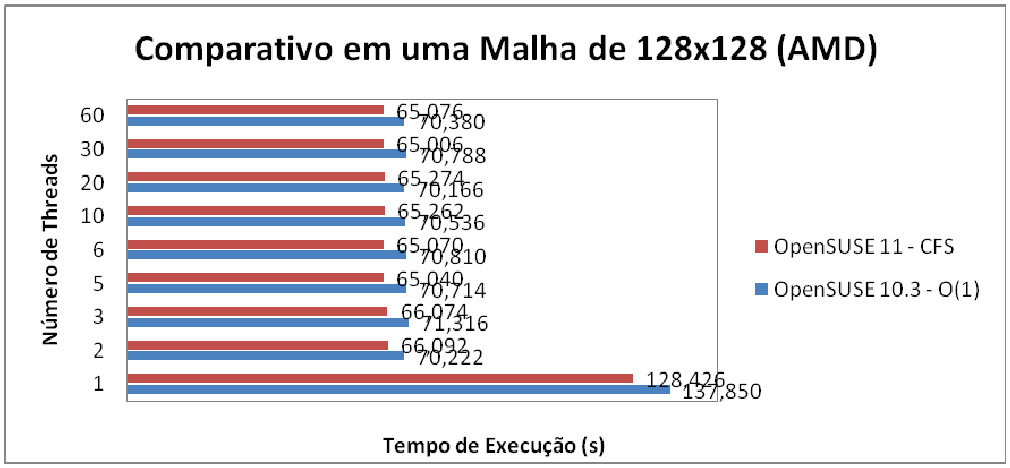
Figura 1 - Tempo de Execução de uma malha 128x128 (Intel) com dois escalonadores.



O segundo estudo de caso também foi realizado em um *notebook*, com a seguinte configuração *AMD Turion 64 X2 Mobile TL-50*, núcleo *Taylor*, 2 x 256 *KB* de *Cache L2*, 1.6 *GHz*, *FSB HyperTransport* 800 *MHz*, 1 *GB DDR2* de memória *RAM*, *OpenSUSE* 11 64 *bits* (*kernel* 2.6.25.5-1.1-*default* e gcc 4.3.1), 1 *GB* de *Swap*. *OpenSUSE* 10.3 64 *bits* (*kernel* 2.6.22.5-31-*default* e gcc 4.2.1). Na Figura 2 observa-se uma redução no tempo de execução do algoritmo utilizando as *threads*, em média cerca

de 48,77% de redução no O(1) e 49,11% de redução no *CFS*. Ainda foi observado um ganho de desempenho de até 8% na utilização do novo escalonador *CFS* ao O(1).

Figura 2 - Tempo de Execução de uma malha 128x128 (AMD) com dois escalonadores.



6. Considerações Finais

O problema abordado neste trabalho constitui-se da paralelização do problema inverso para a resolução do problema da retirada de água do solo pelas raízes de plantas. Foi observado uma importante ganho de desempenho nos tempos de execução em relação ao programa sequencial e ainda uma redução no tempo de execução com o novo escalonador do *GNU/Linux*, o *CFS,* em relação ao O(1). Além disso, mesmo aumentando o número de threads de 2 para 60, o desempenho é pouco alterado. Isto ocorre, pois não há mais núcleos disponíveis para executar novamente em paralelo.

Referências

Borges, Pedro A. P. e Klamt, Vanessa (2006). Modelagem Matemática da retirada de água do solo pelas raízes de plantas. In: XXIX CNMAC - Congresso Nacional de Matemática Aplicada e Computacional. Campinas.

Deives, Crístian; Sabóia, Priscila (2007) "Escalonamento no Linux", http://www.ic.unicamp.br/~islene/2s2007-mo806/slides/MO806_Escalonamento_no_Lin.html, Dezembro.

Lima, Cássio; Torres, Gabriel (2006) "Por Dentro da Arquitetura AMD64", http://www.clubedohardware.com.br/artigos/1213/1, Agosto.

Meira, Marcos V. (2008) "Política de escalonamento de processos em linux", http://www.revista.grupointegrado.br/campodigital/include/getdoc.php?id=206&article=80&mode=pdf, Dezembro.

Molnar, Ingo (2007) "This is the CFS scheduler", http://people.redhat.com/mingo/cfs-scheduler/sched-design-CFS.txt, Dezembro.

Padoin, E. L.; Borges, Pedro A.P.; Dill, Sérgio L; Padilha, Nelson R.. Resolução do problema de absorção da água do solo através da paralelização do problema inverso. In: ERAD 2008 - VIII Escola Regional de Alto Desempenho, 2008, Santa Cruz do Sul.: Evangraf, 2008. v. 1. p. 268-271

Torres, Gabriel; Lima, Cássio (2006) "Por dentro da Microarquitetura Intel Core", http://www.clubedohardware.com.br/artigos/1203, Junho.