Suporte do modelo Split-compute-merge em Anahy Vanilla

Elvio A. A. Viçosa Junior[1], **Gerson Geraldo H. Cavalheiro**[1]

[1]Departamento de Informática – Universidade Federal de Pelotas (UFPel)
Pelotas – RS – Brazil

{evicosa_ifm,gersonc}@ufpel.edu.br

Resumo. *Com a popularização dos processadores multi-core no mercado a indústria foi motivada a produzir uma grande quantidade de processadores com suporte à esta tecnologia, com uma ampla variedade na configuração dos seus recursos, tornando uma árdua tarefa a portabilidade de desempenho entre diferentes configurações. Uma das soluções para diminuir a dificuldade do desenvolvimento de aplicações com desempenho portável entre estas arquiteturas é o uso ferramentas que permitam abstrair a programação da arquitetura alvo, não precisando gerenciar explicitamente os recursos disponíveis pela arquitetura.*

1. Introdução

Durante muito tempo a utilização de programação leve com o objetivo de melhorar o desempenho de aplicações com alto custo computacional foi um recurso utilizado apenas em meios acadêmico ou grandes empresas que possuiam dinheiro para adquirir o caro hardware multiprocessado (SMP - Symmetric Multi Processing) e pessoas especializadas para tirar proveito dos recursos oferecidos por este hardware. Com a popularização dos processadores multicore, a tecnologia que permite a exploração de programação leve ultrapassou a fronteira antes limitada pelo alto custo. Praticamente todos os computadores pessoais produzidos hoje já possuem mais de um núcleo de processamento. Juntamente com a possibilidade de ganho de desempenho pelo uso de processadores multiprocessados, veio a dificuldade de explorar corretamente a grande variedade de configurações destes recursos que hoje estão disponíveis.

É apresentado neste trabalho a implementação do modelo split-compute-merge (SCM) sobre o modelo Anahy Vanilla[1], que tem como objetivo facilitar a utilização de recursos de processadores multiprocessados pelo uso de threads, sobre aplicações que permitem a utilização deste modelo.

Este trabalho esta organizado da seguinte forma: a seção 2 apresenta o modelo Anahy Vanilla. Na seção 3 é abordado o modelo SCM e como ele pode ser utilizado. A seção 4 apresenta um estudo de caso da utilização do SCM juntamente com análises de performance. A conclusão deste trabalho é feita na seção 5.

2. Modelo Anahy Vanilla

Anahy Vanilla [Cavalheiro et al. 2006] é um modelo concebido para programação e execução paralela. Fazendo uso de um núcleo de escalonamento no nível de usuário, ele permite que toda a concorrência definida pela aplicação seja mapeada de forma a

[1]Mais informações em: http://www.anahy.org

não sobrecarregar os recursos disponíveis pelo hardware, permitindo então que a mesma aplicação possa migrar entre diferentes configurações de arquiteturas sem a necessidade de alterações no código. Esta portabilidade é obtida pela utilização de Processadores Virtuais (PV). Os PVs são threads de sistema que são criadas durante o inicio da execução de um programa usando o modelo Anahy Vanilla, e o sistema operacional fica responsável pelo escalonamento destas threads. Pode-se afirmar que a concorrência real é limitada pelo número de PVs criados durante o início da execução e não pelo número de threads definidas pela aplicação, podendo então definir o paralelismo da aplicação baseada nas suas características, idependentemente do hardware alvo da aplicação, tornando a aplicação adaptável à diferentes configurações. A organização do modelo Anahy Vanilla pode ser observado na Figura 1.

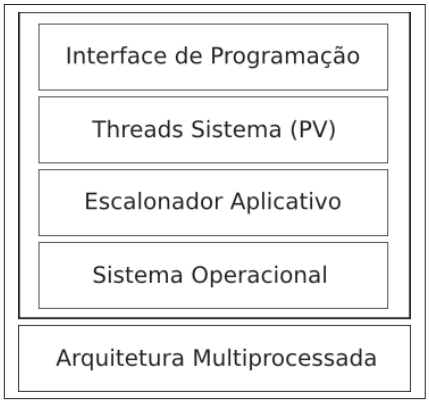


Figure 1. Camadas do modelo Anahy Vanilla

O ambiente Anahy Vanilla oferece uma interface de programação, permitindo que novos fluxos concorrentes sejam sejam criados
sincronizados utilizando primitivas do tipo split
join. Esta interface segue o padrão POSIX [Drepper and Molnar 2005] para threads. Um exemplo da sua utilização é demonstrado no trecho de código mostrado na Figura 2.

```
void *inc(void *in) {
    int *result=malloc(sizeof(int));  *result = *(int *)in + 1;
    return result;
}

int main(int argc, char **argv) {
    int *valor = malloc(sizeof(int)); int *valor_plus_one;
    athread_t thread; *valor = 10;
    athread_create(thread, NULL, inc, (void*) valor);
    athread_join(thread, valor_plus_one);
    printf("%d + 1 = %d", *valor, *valor_plus_one);
}
```

Figure 2. Exemplo de criação e sincronizações de fluxo no modelo Anahy Vanilla

Além de possuir como característica portabilidade de desempenho entre diferentes configurações de arquitetura, o ambiente Anahy Vanilla também possue a característica

de tirar proveito da localidade espacial do uso da memória durante a execução. Todos os fluxos criados (e ativos) são mantidos em um grafo organizado pela dependência entre as threads e diferentes algoritmos são utilizados para examinar o grafo e definir a ordem de execução, sendo por padrão utilizado o algoritmo que executa as threads mais novas (base do grafo) inicialmente. A Figura 3 mostra o grafo que mantém as threads criadas no modelo Anahy Vanilla.

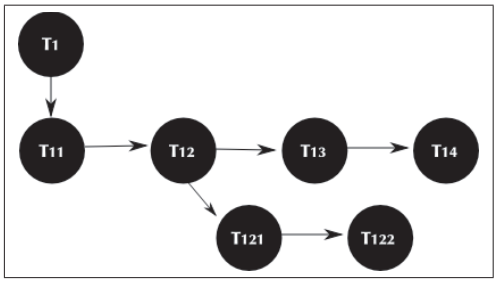


Figure 3. Grafo do núcleo de gerenciamento do modelo Anahy Vanilla

3. Split, Compute and Merge (SCM)

Existe uma grande parcela de problemas que possuem um comportamento em comum: uma grande quantidade de dados pode ser dividida, computada independentemente e o resultado final obtido pela combinação destes resultados. Tendo em vista esta grande quantidade de aplicações deste tipo, foi incorporado o SCM ao modelo Anahy Vanilla, permitindo que o programador possa utilizá-lo com poucas configurações.

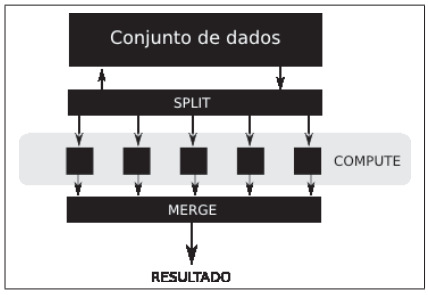


Figure 4. Esquema da execução do modelo Split-Compute-Merge

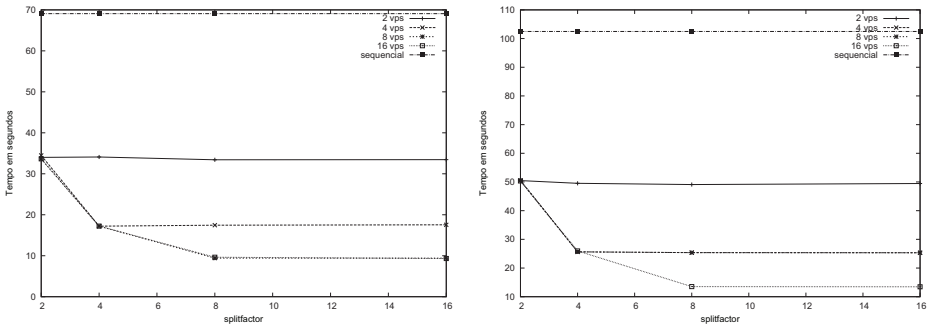
A Figura 4 mostra como a partir de um conjunto global de dados, estes dados são quebrados em pequenos conjuntos, que serão passados de forma transparente à threads que irão computar os resultados parciais concorrentemente, retornando-o ao final. A função merge fica responsável por organizar estas informações da forma que melhor representa as características da aplicação e retornar o resultado final.

Um ponto importante a ser notado é que o programador em momento algum precisa se preocupar com a criação
sincronização de threads. Apenas foi definido o número de partes que o bloco de memória será dividido (splitfactor) e o conteúdo das funções responsráveis pela divisão do conjunto de dados, computação dos resultados parciais e combinação dos resultados parciais, sendo o ambiente de execução responsável pela definição da concorrência da aplicação.

A aplicação do modelo SCM não só torna a aplicação mais simples de ser desenvolvida, uma vez que as definições feitas são desenvolvidas como programação seqüêncial, más também torna muito mais rápida a resolução de problemas que utilizam este modelo.

4. A aplicação SCM.Zip

A fim de validar e testar o modelo SCM, foi desenvolvido como estudo de caso uma aplicação [2] que aplica um algoritmo de compactação de forma paralela em um determinado arquivo. A partir de um arquivo informado pela linha de comando, a aplicação divide o conteúdo deste arquivo em blocos de tamanhos iguais e os compacta concorrentemente, utilizado a biblioteca zlib[3] para esta compactação. Este estudo de caso foi executado em um ambiente GNU
Linux com 16 processadores (com acesso a memória não uniforme - NUMA) e um total de 8GB de memória. Os arquivos a serem compactados possuem respectivamente 410 e 610 Mb e foram utilizados em todos os casos valores de splitfactor = 2, 4, 6 e 8.



(a) Arquivo com 410Mb (b) Arquivo com 610Mb

Figure 5. Gráficos da execução das versões paralelas e versão seqüêncial usando o modelo SCM

5. Conclusão

Os resultados obtidos pelo estudo de caso mostram um grande ganho de desempenho em todas as versões paralelas, quando comparado aos resultados obtidos durante a execução seqüêncial da aplicação. Pode ser verificado pelos gráficos, que é possível escalar o desempenho a medida que há uma variação no número de processadores virtuais, sendo os melhores resultados obtidos quando o número de PVs se aproxima aos recursos reais de paralelismo da arquitetura utilizada.

References

Cavalheiro, G., Paschoal, L., Cardozo, M., and Cordeiro, O. (2006). Anahy: a programming environment for cluster computing. *Proc. of the VII International Meeting on High Performance Computing for Computational Science*.

Drepper, U. and Molnar, I. (2005). The native posix thread library for linux. *http://people.redhat.com/drepper/nptl-design.pdf*.

[2] Código disponível em: http://github.com/elvio/zipper/tree
[3] http://www.zlib.net