Dequeues auto balanceáveis para algoritmos branch and bound em MPI-1

Stéfano Drimon Kurz Mór, Nicolas Maillard

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{sdkmor, nicolas}@inf.ufrgs.br

1. Introdução

A *Message-Passing Interface* (MPI) é o padrão em computação paralela com troca de mensagens. Por questões de portabilidade, MPI não provê uma maneira eficiente de distribuição de carga de trabalho gerada em tempo de execução aos computadores participantes. Isso é um problema, em especial, quando se consideram algoritmos do tipo *branch and bound* (B&B), que podem produzir, intrinsecamente, grande desbalanceamento de carga de trabalho entre os recursos disponíveis.

Propõe-se, então, uma biblioteca sobre MPI-1 que possa oferecer suporte à criação e escalonamento eficiente de tarefas criadas dinamicamente em algoritmos B&B. O nome dessa biblioteca é RAWSDM (*Randomized Work-Stealing for Distributed Memory*).

Algoritmos paralelos que executam em forma de árvore – como B&B – não são novos. Uma das primeiras contribuições sobre o assunto é vista em [Wu e Kung 1991], que apresenta a análise dos limites assintóticos inferiores no escalonamento de nós da árvore em P processadores (totalmente) interconectados. Esse limite é alcançado usando-se um algoritmo baseado na distribuição de *tokens*, cujo custo de comunicação é $P \cdot h \cdot \delta$ (sendo h a altura da árvore e δ o maior grau). Outros aspectos desse escalonamento são abordados em diferentes contribuições, como em [Nieuwpoort et al. 2001] e [Narlikar e Blelloch 1999].

2. Programando com RaWSDM

Essencialmente, RAWSDM provê ao programador uma estrutura de dados do tipo *deque* (*double ended queue*) na qual se podem retirar, processar e reinserir tarefas.

A programação com RAWSDM é baseada no seguinte ciclo:

1. O programa decompõe uma estrutura de dados que representa uma tarefa em subestruturas (subtarefas).
2. Essas subtarefas são postas no topo da *deque* e uma lista de dependências entre tarefas é atualizada com as novas dependências geradas pelas subtarefas.
3. O programa retira uma subtarefa do fundo da *deque*. Vá para (1).

Esse ciclo prossegue *ad infinitum* até que uma detecção distribuída de término execute com sucesso, o que só acontece quando a árvore é toda expandida.

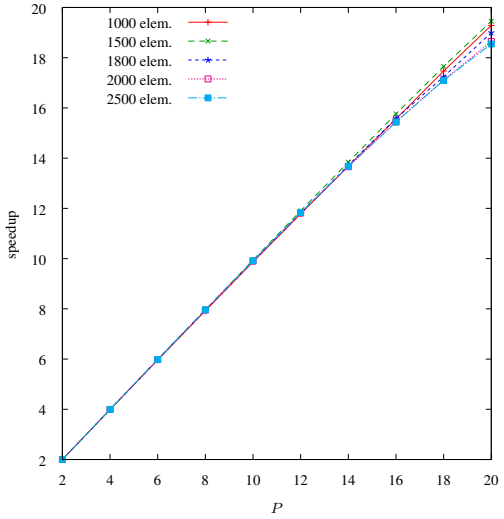
Uma camada de *software*, invisível ao programador, mantém todas as *deque* com o mesmo número de tarefas, distribuindo-as de maneira transparente e dinâmica, via rede. O algoritmo utilizado é o roubo da tarefas com critério aleatório, demonstrado em [Blumofe e Leiserson 1994] ser (probabilisticamente) ótimo.

O *overhead* de escalonamento varia logaritmicamente com o trabalho total, sendo próximo ao melhor caso assintótico no tempo de execução $\frac{T_1}{P} + \mathcal{O}(\log_\delta T_1)$, para uma árvore com T_1 nós. A memória ocupada também obedece limites assintóticos ótimos.

3. Experimentos e Resultados

A aplicação de teste utilizada é uma solução B&B para o Problema da Mochila (versão ilimitada), revisado em [Keller et al. 2005]. Testes foram executados com MPI-1.2 (LAM-MPI 7). Os resultados, em termos de *speedup*, podem ser observados na Fig. 1.

Figura 1. *Speedup* do RAWSDM.



Resultados em outros testes mostram um ganho possível de até 80% em relação a uma implementação B&B que usa *round-robin* para o escalonamento, como em várias implementações clássicas.

4. Conclusões

Esses resultados foram aceitos para publicação em uma edição especial do *International Journal on Parallel Systems Architectures*, ainda a ser lançada. Em nossa opinião eles mostram a possibilidade de implementação eficiente de uma camada de criação de tarefas dinâmicas que as multiplexa em processos MPI-1, criados estaticamente, com *overhead* mínimo. Esse tipo de suporte é essencial em várias aplicações, em especial as resolvem problemas de Pesquisa Operacional, tipicamente solucionados através de algoritmos B&B.

Referências

Blumofe, R. D. e Leiserson, C. E. (1994). Scheduling multithreaded computations by work stealing. Em *In Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS)*, páginas 356–368.

Keller, H., Pferschy, U., and Pisinger, D. (2005). *Knapsack Problems*. Springer.

Narlikar, G. J. e Blelloch, G. E. (1999). Space-efficient scheduling of nested parallelism. *ACM Trans. Program. Lang. Syst.*, 21(1):138–173.

Nieuwpoort, R., Kielmann, T., e Bal, H. E. (2001). Efficient load balancing for wide-area divide-and-conquer applications. Em *PPoPP '01: Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming*, páginas 34–43, Nova Iorque, NY, EUA. ACM.

Wu, I. e Kung, H. T. (1991). Communication complexity for parallel divide-and-conquer. Em *In Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, páginas 151–162.