Quantificando o impacto do rastreamento em aplicações paral1elas OpenMP baseadas em tarefas

Rayan Raddatz de Matos, Lucas Mello Schnorr

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)

Resumo. *Para entender o que acontece durante a execução de uma aplicação paralela baseada em tarefas é necessário instrumentá-la e analisar os dados (rastros) gerados. Entretanto ao rastrear a execução estamos alterando o ambiente e comportamento originais da aplicação. Este trabalho foca em quantificar o impacto causado pelo rastreamento das tarefas no tempo de execução de uma aplicação paralela e quais fatores influenciam esse impacto. Os resultados demonstram que o número de threads e a quantidade de tarefas realizadas são importantes fatores de impacto no tempo consumido pelo rastreador.*

1. Introdução

O aumento constante no desempenho computacional que pode ser observado no mundo traz também um aumento na complexidade observada nas máquinas atuais, com aplicações cada vez mais complexas e paralelizadas. Assim, uma boa compreensão do comportamento da execução de um programa é essencial para avaliações tanto sobre o código executado quanto sobre a máquina que o executa.

Nesse cenário, a técnica de rastreamento se torna amplamente utilizada para entender a execução, consistindo na acoplação de código adicional ao programa. Tal código registra os eventos ocorridos e tarefas efetuadas durante a execução da aplicação, com o objetivo de dar ao usuário conhecimento sobre como o programa foi realizado. Entretanto, usar um rastreador impacta o ambiente e o desempenho da aplicação, modificando o curso padrão da execução. Este trabalho, portanto, visa mensurar o tamanho desse impacto e identificar os fatores que o amplificam. Para isso, a aplicação utilizada foi uma fatoração QR por blocos, implementada de forma paralela utilizando OpenMP com tarefas e rastreada utilizando um plugin que segue a API de callbacks do OMPT, executada com diferentes tamanhos de problema, quantidade de threads e configurações do rastreador.

Na Seção 2 são destacados alguns trabalhos relacionados. A Seção 3 aborda o processo de rastreamento e descreve brevemente a aplicação utilizada e os motivos para sua escolha. A Seção 4 fala sobre a metodologia utilizada para os experimentos. Por fim, são mostrados os resultados na Seção 5 e uma conclusão é dada na Seção 6.

2. Trabalhos Relacionados

Diferentes estudos buscam compreender o comportamento de aplicações baseadas em tarefas, analisar seu desempenho e entender fatores de impacto durante a sua execução. Em [Miletto et al. 2020] é analisado o impacto de diferentes *runtimes* na execução de uma aplicação baseada em tarefas, concluindo que alguns apresentam um desempenho abaixo do esperado quando o grão de trabalho é pequeno. Já em [Laurino and Pinto 2023], são feitas análises e comparações do impacto ao utilizar diferentes interfaces para o rastreamento de três aplicações, concluindo que, dependendo da forma à qual é realizado, o rastreamento pode ocasionar grande impacto no desempenho da aplicação.

Diferente dos trabalhos mencionados, este artigo foca em analisar o impacto no rastreamento de uma mesma aplicação paralela com diferentes configurações e escalas de problema, buscando analisar o desempenho com intensidades de tarefas distintas e entender os fatores que aumentam o tempo consumido pelo rastreador.

3. Ferramentas de rastreamento e a aplicação rastreada

Essa seção aborda em mais detalhes o rastreamento de uma aplicação. Além disso, também descreve a aplicação utilizada, que consiste em uma fatoração QR por blocos.

3.1. Rastreamento de uma aplicação

A ferramenta de programação paralela em memória compartilhada **Open Multi-Processing** (OpenMP) [Chandra 2001] utiliza diretivas para criar tarefas e gerenciar o paralelismo da aplicação. Enquanto isso, a ferramenta **OpenMP Tools Interface** (OMPT) [Eichenberger 2013] pode ser utilizada para conseguir informações sobre a execução dessas tarefas. A interface OMPT permite que o usuário defina funções que serão chamadas quando eventos referentes às threads forem acionados durante a execução do programa paralelizado com OpenMP, registrando aspectos como o estado atual da thread, a tarefa sendo realizada naquele instante, marcações de início e fim de tarefas e outros.

3.2. Aplicação rastreada: Fatoração QR por blocos e com tarefas

Dada uma matriz A de dimensões $m \times n$ podemos, através da fatoração QR, transforma-lá em duas matrizes Q e R, sendo Q uma matriz ortogonal de dimensões $m \times m$ e R uma matriz triangular superior de dimensões $m \times n$. Essa fatoração é utilizada para resolver diversos problemas descritos por equações lineares, além de ser muito importante na resolução do problema dos mínimos quadrados. Dentre as várias opções de implementação para a fatoração QR, a que utiliza o método de reflexões de Householder [Householder 1958] é uma das mais vantajosas para ambientes de alto desempenho por permitir a realização da fatoração por blocos com maior grau de paralelismo mesmo realizando mais operações de ponto-flutuante. Sendo assim, ao invés de computar a matriz de tamanho $m \times n$, computamos diversas submatrizes de dimensão $nb \times nb$, onde nb é o tamanho do bloco.

Com essa estratégia de fatoração QR por blocos é possível computar diversos blocos paralelamente e, dependendo da razão entre o tamanho da matriz e o tamanho do bloco, podemos ter mais ou menos paralelismo presente durante a execução. Assim, essa aplicação se torna uma escolha sensata para os objetivos deste trabalho, tornando possível a análise de múltiplos níveis de paralelismo em uma mesma aplicação.

4. Metodologia experimental

Os experimentos foram realizados com matrizes de tamanhos entre 1024, 2048 e 4096 e tamanhos de bloco entre 32, 64, 128 e 256 em três máquinas distintas, detalhadas na Tabela 1. Buscando avaliar o efeito da tecnologia de hyper-threading presente nos processadores na intrusão do rastreamento, as execuções se diferenciam ainda naquelas que usam todos os cores daquelas que usam apenas os cores físicos. Enfim, para garantir que as execuções encerrassem, evitando a falta de paralelismo ou o excesso de tarefas, foram utilizadas somente combinações em que a razão entre o tamanho da matriz e do bloco fosse maior ou igual a 8 e menor ou igual a 64.

Tabela 1. Especificações de hardware das máquinas utilizadas.

Nome	CPU	RAM
Draco	2 x Intel(R) Xeon(R) E5-2640 v2, 2.00 GHz, 32 ths, 16 cores	64 GB DDR3
Cei	2 x Intel(R) Xeon(R) Silver 4116, 2.10 GHz, 48 ths, 24 cores	96 GB DDR4
Hype	2 x Intel(R) Xeon(R) E5-2650 v3, 2.30 GHz, 40 ths, 20 cores	128 GB DDR4

Para cada combinação de matriz, bloco e número de threads foram realizadas três tipos de execuções, rastreando eventos referentes às tarefas em cada thread. Exemplos de eventos rastreados são início e fim de uma thread ou de uma tarefa e troca de tarefa. Os tipos de execução constam a seguir, cada um mudando o código interno do rastreador:

- **Normal/Sem rastreamento:** Execução sem o uso a ferramenta OMPT, tendo assim essa execução como base do tempo de execução da aplicação.
- **Rastreamento vazio:** Execução que utiliza a interface OMPT para criar as chamadas de funções para eventos, entretanto os conteúdos das funções é nulo, contabilizando apenas a sobrecarga de chamada para interface OMPT.
- **Rastreamento com escrita em arquivo:** Execução que utiliza da interface OMPT para criar chamadas para eventos. Assim que um evento ocorre salva um registro do evento em um arquivo dedicado para a thread atual.

Além disso, para lidar com a variabilidade do experimento, cada combinação de tamanho de matriz, tamanho de bloco, número de threads e código do rastreador foi executada 30 vezes, sendo considerada a média aritmética do tempo de cada rastreamento. Consideramos que o tempo de execução respeita uma distribuição gaussiana, o erro foi calculado assumindo um intervalo de confiança de 99.95%.

5. Resultados

Comparando o resultado das execuções sem e com rastreamento, foi notado que o impacto do rastreamento é mínimo em alguns casos. Entretanto, casos em que a razão entre o tamanho da matriz e o tamanho do bloco é grande ocasionaram um alto número de tarefas e, consequentemente, o impacto no tempo consumido pelo rastreador também foi mais prevalente. Além do número de tarefas realizadas, um alto número de threads utilizadas também amplifica o tempo consumido pelo rastreador pela necessidade de rastrear o comportamento de cada uma. A Figura 1 compara o resultado de uma execução com e sem hyper-threading em uma configuração que gera 89.444 tarefas.

É possível notar que tivemos um aumento no tempo gasto pelo rastreador tanto na máquina Cei quanto na máquina Hype ao utilizar o hyper-threading. Como exemplo, ao rastrear a execução do problema utilizando sem hyper-threading na máquina Hype, o impacto no tempo foi de 3.85% em relação à execução sem o rastreador, enquanto ao utilizar o hyper-threading, esse impacto foi de 10.23%, um aumento de 6.38% no tempo gasto pelo rastreador que se deve ao incremento no número de threads. Esse acréscimo também indica a ineficácia do hyper-threading nesta aplicação baseada em CPU, gerando uma maior sobrecarga devido à concorrência de threads que efetuam muitos cálculos.

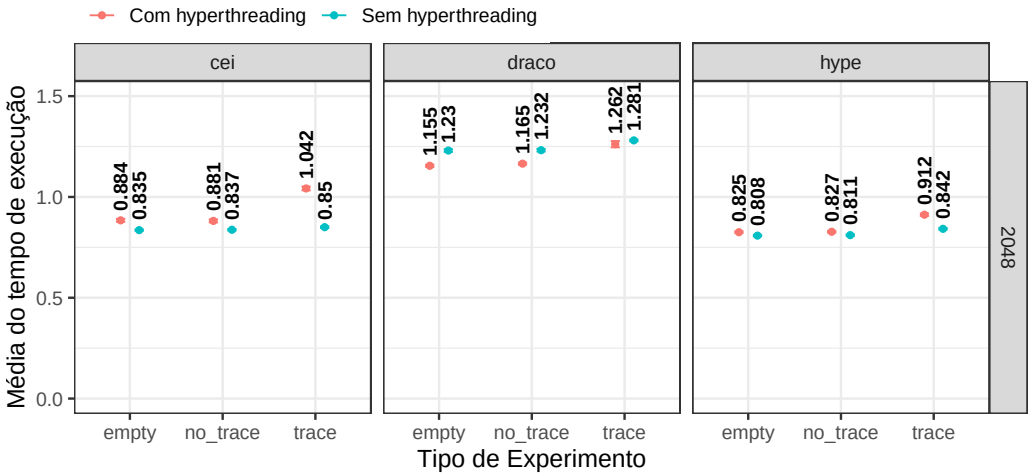

Figura 1. Média dos resultados para tamanho de matriz de 2048 e bloco de 32.

Já as execuções com o rastreamento vazio não mostraram diferenças relevantes no tempo em relação à execução sem o rastreador, mostrando que a interface OMPT tem uma baixa intrusão por si só, sendo o código do rastreador responsável pelo impacto causado.

6. Conclusão

O rastreamento é uma importante prática para entender a execução de uma aplicação paralela. Entretanto, em ambientes com alto nível de paralelismo, com grandes quantidades de tarefas e threads, o rastreio da aplicação pode acabar gerando um grande impacto no desempenho. Sendo assim, é importante ao usuário saber balancear o uso de paralelismo, buscando um rastreamento mais eficiente e com menos intrusão. Trabalhos futuros visam analisar outros códigos para o rastreador e fazer uso da biblioteca LibRastro [da Silva and de Oliveira Stein 2002].

Referências

Chandra, R. (2001). *Parallel Programming in OpenMP.* Academic Press.

da Silva, G. J. and de Oliveira Stein, B. (2002). Uma biblioteca genérica de geração de rastros de execução para visualização de programas. *Anais do I Simpósio de Informática da Região Centro*.

Eichenberger, Alexandre E., e. a. (2013). Ompt: An openmp tools application programming interface for performance analysis. *OpenMP in the Era of Low Power Devices and Accelerators: 9th International Workshop on OpenMP*, page 171 – 185.

Householder, A. S. (1958). Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM*, page 339 – 342.

Laurino, J. V. and Pinto, V. G. (2023). Análise do overhead em aplicações paralelas openmp utilizando ompt e score-p. *Anais da XXIII Escola Regional de Alto Desempenho da Região Sul.*

Miletto, M. C., Schnorr, L. M., Pinto, V. G., and da Silva, H. C. P. (2020). Análise da influência do runtime openmp no desempenho de aplicação com tarefas. *Anais da XX Escola Regional de Alto Desempenho da Região Sul*.