Análise do overhead em aplicações paralelas OpenMP utilizando OMPT e Score-P

João Vitor Laurino*, Vinícius Garcia Pinto

[1]C3 - Centro de Ciências Computacionais – Universidade Federal do Rio Grande (FURG)

{joaos.laurino, vinicius.pinto}@furg.br

Resumo. *Com o aumento constante de aplicações paralelas, tornou-se necessário a utilização de ferramentas que permitam uma melhor visualização do comportamento destas aplicações no intuito de facilitar a otimização e obter o melhor desempenho possível. O foco deste trabalho é analisar o impacto (overhead) causado por ferramentas de rastreamento de aplicações OpenMP utilizando OMPT e Score-P.*

1. Introdução

O paralelismo em aplicações de alto desempenho tornou-se indispensável em diversas áreas científicas quando almejamos a exploração eficiente de recursos computacionais. No entanto, a alta escalabilidade e o indeterminismo gerado na execução devido ao uso de diversos nós de processamento acabam gerando uma alta complexidade. Sendo assim, é fundamental avaliar os fatores que limitam o desempenho de uma aplicação paralela.

Existem diversas estratégias para controlar a complexidade das aplicações paralelas, através da utilização de ferramentas como o OpenMP que permite gerar código *multithreading* para um único nó computacional, ou também por meio de bibliotecas que permitam a comunicação entre processos de forma mais eficiente, como a troca de mensagens oferecida pela especificação *Message Passing Interface* (MPI) [Schnorr 2014]. Com a utilização destas ferramentas, o desenvolvedor busca projetar as aplicações de forma com que os recursos sejam usados ao máximo para atingir o alto desempenho desejado.

O sucesso da otimização de uma aplicação de alto desempenho é determinado pela redução do seu tempo de execução. Para alcançarmos esta melhoria, é necessário que, com a aplicação já pronta, seja efetuada uma análise de desempenho. A análise da aplicação consiste em duas etapas: a coleta de dados, que consiste em obter dados durante a execução da aplicação e registrar informações importantes em relação ao comportamento da aplicação. Para então efetuar uma análise destes dados obtidos, a fim de identificar os problemas que geram redução no desempenho da aplicação.

O rastreamento de aplicações consiste em coletar informações sobre o comportamento de um programa durante a sua execução. Permitindo recriar, em um momento posterior, a execução da aplicação paralela, identificando quando cada etapa começou e terminou e onde foi realizada. Este artigo tem como objetivo analisar o *overhead* causado por ferramentas de rastreamento de aplicações paralelas como o OMPT *(OpenMP Tools Interface)* e o Score-P, ou seja, verificar qual o impacto que pode ser gerado na aplicação ao utilizar uma destas ferramentas para a coleta de dados.

*Trabalho parcialmente financiado pelo PROBIC/FAPERGS

2. OpenMP

O OpenMP (*Open Multi-Processing*) é uma API utilizada para a programação paralela explícita e funciona através do paralelismo em memória compartilhada, baseado no modelo fork-join [OpenMP 2020]. Tal modelo que consiste em atuar por meio de uma *thread* mestre que inicia a execução e que posteriormente gera *sub-threads* de trabalho para executar as tarefas em paralelo.

Versões mais recentes do OpenMP permitem a programação paralela com as diretivas orientadas a tarefas ou OpenMP Tasks. A principal diretiva para definirmos uma tarefa é dada por `#pragma omp task` antes de um bloco de código o qual receberá esta tarefa. Com isso, é criada uma nova tarefa que poderá ser executada em paralelo junto com o código que está fora da região definida inicialmente para ser a tarefa ou com outras tarefas criadas posteriormente. Já a diretiva `#pragma omp taskwait` serve para colocarmos uma determinada tarefa em estado `wait`, ou seja, para que ela fique esperando até que as outras tarefas filhas geradas pela principal terminem a sua execução. Por fim podemos também adicionar a cláusula `depend` que serve para informar a dependência de uma variável em uma tarefa, permitindo sincronizações mais finas.

Já o rastreamento de uma aplicação OpenMP pode ser obtido tanto de maneira nativa utilizando a interface *OpenMP Tools Interface* (OMPT) quanto por meio do uso de ferramentas externas como o Score-P. Entre as funcionalidades suportadas pelo OMPT está o uso de *callbacks* que permitem capturar eventos da execução OpenMP.

3. Metodologia

Foram selecionadas três aplicações paralelas para executar os experimentos. Os códigos utilizados foram retirados do minicurso [Nesi et al. 2021]. As aplicações são:

- **Mergesort** - Aplicação baseada no modelo divisão e conquista através da ordenação por mistura trabalhando de forma recursiva. Esta aplicação utiliza somente as diretivas `task` e `taskwait`;
- **Gauss-Seidel** - Algoritmo da suavização de Gauss-Seidel que consiste em efetuar a otimização de laços aninhados através da utilização de matrizes. Neste algoritmo é utilizado a primitiva `task` e a cláusula `depend`;
- **Cholesky** - A Fatoração de Cholesky consiste em fatorar uma matriz simétrica positiva-definida A em uma matriz triangular L e sua transposta L^T ($A = LL^T$). O algoritmo utiliza matrizes em 2D mapeadas para vetor 1D onde também é utilizado a primitiva `task` e a cláusula `depend`.

Foram executados três testes com cada uma das aplicações para verificarmos a discrepância no tempo de execução (*overhead*) entre elas. Para obter a coleta de dados, inicialmente, foram feitos os testes somente utilizando as aplicações com OpenMP sem rastreamento para usar como parâmetro de comparação. Logo após foram efetuados testes habilitando o rastremento das três aplicações utilizando as ferramentas OMPT e Score-P.

3.1. Ambiente de Validação

Todos os experimentos foram realizados no *cluster* da Universidade Federal do Rio Grande (FURG), utilizando um nó de cálculo reservado para uso privado, onde todos os testes foram realizados utilizando 8 *threads*. O cluster possui 64 GB de memória RAM

DDR4 e dois processadores *Intel Xeon E5-2640 v3* que operam na frequência de 2.60GHz, cada um possui 8 núcleos físicos e 16 *threads*, totalizando 16 núcleos físicos e 32 *threads*. O Sistema Operacional utilizado no *cluster* é o CentOS 7.3, kernel Linux 3.10.0.

Os códigos utilizados para o experimento são todos escritos em C, compilados `gcc` 12.1 e `LLVM/clang` 11, através do *runtime* `libomp`. Para a ferramenta Score-P foi utilizada a versão 7.1. Foi utilizado a diretiva de otimização `-O3` em todos os testes.

3.2. Parâmetros de Entrada

Foi escolhido um tamanho de entrada fixo para cada algoritmo para verificarmos o ganho de desempenho em relação aos três casos (sem rastreamento, com rastreamento via OMPT e via Score-P). Os tamanhos escolhidos para cada aplicação são indicados a seguir:

- **Mergesort** - Para os três casos foi utilizado um vetor de inteiros com 600.000 elementos;
- **Gauss** - Foram utilizados os valores 3000 (Tamanho N da matriz), 10 (Número de iterações de Gauss Seidel), 500 (Tamanho do bloco de tarefas), 10 (Tamanho interno do bloco de tarefas);
- **Cholesky** - Foi utilizado como entrada uma matriz de 5000×5000 (25.000.000 de elementos) e ordem do bloco 250.

4. Resultados

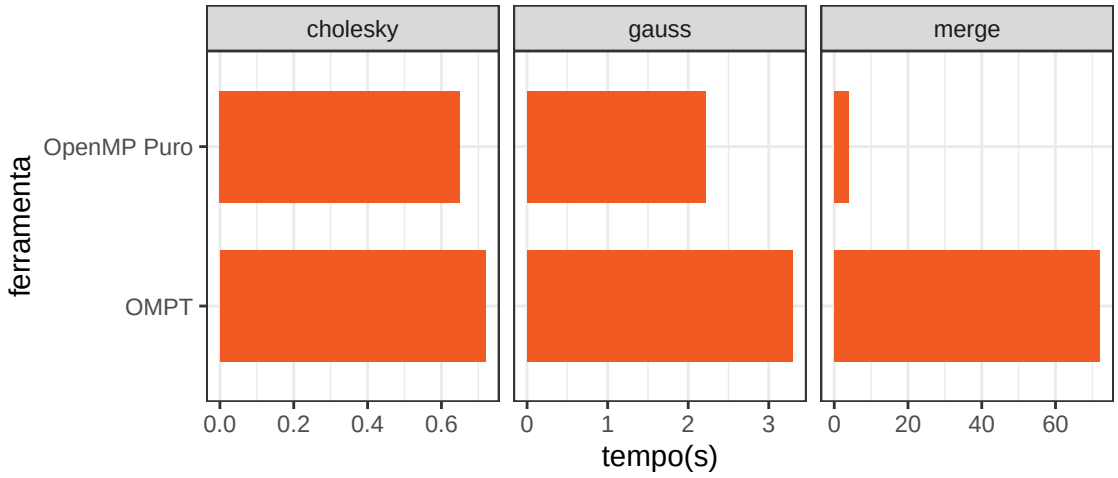

Figura 1. Comparação do tempo de execução das três aplicações paralelas.

A Figura 1 apresenta os resultados obtidos ao executar as três aplicações utilizando OpenMP puro e com OMPT. Analisando os gráficos é possível identificar que o rastreamento pode sim causar um *overhead* sobre a aplicação que está sendo executada, onde este impacto é definido pela forma na qual a aplicação foi desenvolvida. Nos casos apresentados, verificamos que no Mergesort temos um acréscimo no tempo de execução muito maior, devido ao emprego de recursividade, já nos outros dois casos o *overhead* causado foi menos discrepante em comparação com a execução sem rastramento.

Como é possível visualizar não foi indicado os valores ao executar as aplicações com o rastreamento da ferramenta Score-P. Nos três casos testados, utilizando o valor de

memória padrão para rastreamento do Score-P, as aplicações apresentaram falha por falta de memória, não efetuando o processo de rastreamento. Este fato sugere que a ferramenta Score-P não lida bem com aplicações que geram um número excessivo de tarefas.

Na Figura 2, é apresentado um exemplo de visualização dos dados de rastreamento obtidos durante a execução da suavização de Gauss-Seidel com 8 *threads* usando OMPT. Este gráfico mostra a qual iteração pertencem as tarefas em execução em uma determinada *thread* em um dado momento. Este tipo de visualização, auxilia na identificação de eventuais gargalos que possam estar limitando o desempenho da aplicação.

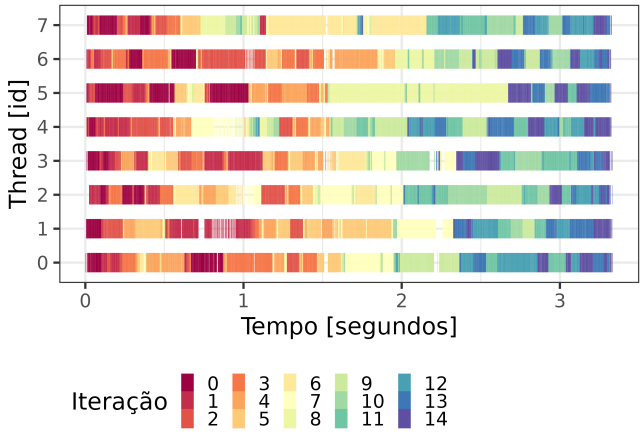

Figura 2. Rastro da execução da aplicação Gauss Seidel obtido via OMPT.

5. Conclusão

A partir dos resultados obtidos com este trabalho, foi possível concluir que o rastreamento é uma ferramenta necessária e útil no processo de análise de desempenho de aplicações paralelas, porém devem ser utilizadas com cautela, visto que, dependendo da forma com a qual ela será aplicada, pode causar um grande impacto no desempenho da aplicação que está sendo testada.

Em trabalhos futuros, pretende-se realizar testes com novas ferramentas de rastreamento, no intuito de obtermos informações mais precisas a respeito da qualidade do rastreamento obtido e do respectivo *overhead* nas diferentes aplicações paralelas. Outro aspecto ser explorado é a influência da quantidade de tarefas geradas no *overhead* adicionado ao empregar o rastreamento. Também está previsa a exploração de outras configurações que possam viabilizar o uso da ferramenta Score-P.

Referências

Nesi, L. L., Miletto, M. C., Pinto, V. G., ; Schnorr, L. M. (2021). Desenvolvimento de aplicações baseadas em tarefas com openmp tasks. In *Minicursos da XXI Escola Regional de Alto Desempenho da Região Sul.*, pages 131–139, Porto Alegre, RS, Brasil.

OpenMP (2020). The OpenMP API specification for parallel programming. [Online; acessado em março, 01 2023. https://www.openmp.org].

Schnorr, L. M. (2014). Análise de desempenho de programas paralelos. In *Anais da XIV Escola Regional de Alto Desempenho da Região Sul*, pages 2–3, Porto Alegre, RS, Brasil.