Supercomputação
Entrega - Atividade 13
Arthur Tamm

Introdução

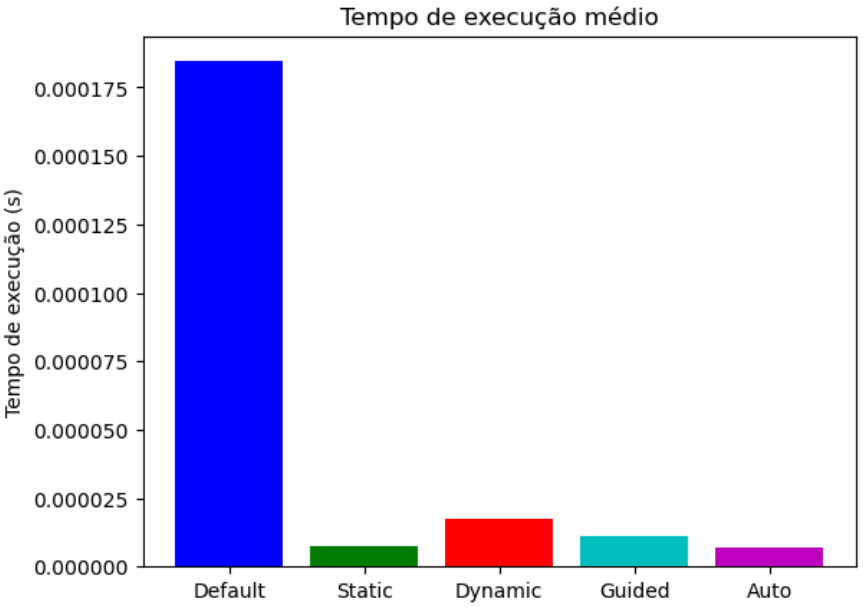
Este relatório explora o uso de schedulers no OpenMP e a paralelização de códigos recursivos em sistemas de memória compartilhada, analisando o impacto de diferentes estratégias, como **static, dynamic, guided, auto e runtime**, no desempenho de loops paralelos. Além disso, investigamos a paralelização de um cálculo recursivo de Pi utilizando as diretivas parallel for e task, comparando os tempos de execução. Também discutimos a manipulação de efeitos colaterais ao modificar vetores compartilhados entre threads, avaliando o uso de regiões críticas e técnicas de pré-alocação de memória para otimizar o desempenho.

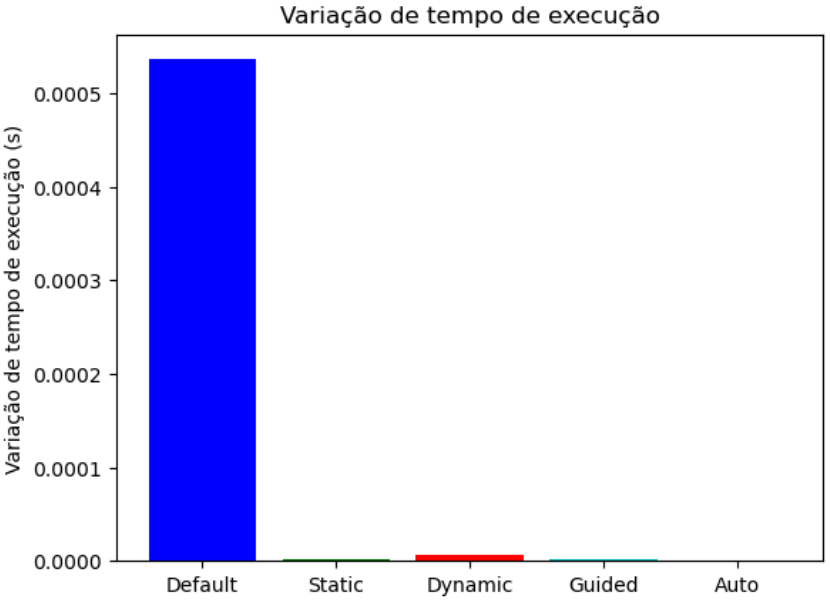
Schedules

Com base nos resultados obtidos, o scheduler **auto** apresentou o menor tempo médio de execução, com 7.03767e-06 segundos. Esse comportamento é esperado, pois o scheduler **auto** permite que o compilador ou a implementação de OpenMP escolham a melhor estratégia de distribuição de iterações, resultando em uma distribuição otimizada para o ambiente de execução específico.

No entanto, o scheduler **Default** apresentou variações significativas entre as execuções, com uma variação de 0.000536189 segundos. Isso pode ser explicado pela natureza adaptativa do scheduler padrão, que pode depender das condições do sistema, como carga atual de threads, o que leva a flutuações na performance.

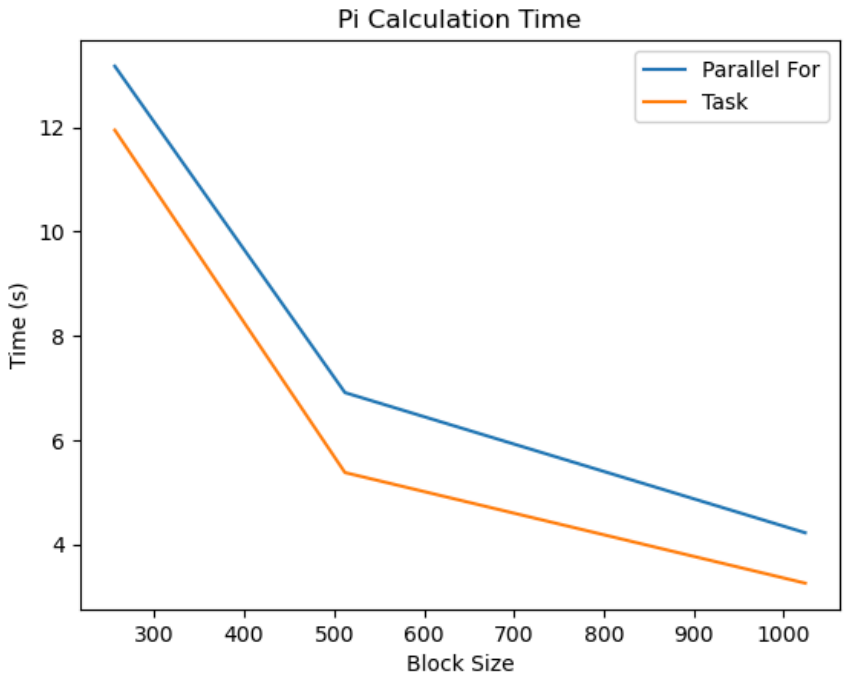
A carga de dados e o balanceamento foram fatores que influenciaram o comportamento dos schedulers. O **static**, por exemplo, apresentou tempos mais consistentes e baixos (7.31967e-06 segundos), uma vez que divide igualmente as iterações entre as threads no início da execução, reduzindo a sobrecarga gerencial. Em resumo, as características do trabalho, como o tamanho da carga e a uniformidade das iterações, influenciam diretamente a eficiência de cada scheduler, com o **auto** sendo o mais eficiente neste contexto, e o **dynamic** e **default** sendo mais suscetíveis a variações.





Cálculo do PI

Ao paralelizar o cálculo recursivo de Pi utilizando duas abordagens distintas, parallel for e tasks, observamos diferenças significativas nos tempos de execução e no impacto de parâmetros como o valor de MIN_BLK.



A abordagem com tasks apresentou melhor desempenho em todos os cenários testados. Para o valor de MIN_BLK igual a 1024, o tempo de execução foi de 3.2537 segundos, comparado a 4.2208 segundos com a abordagem parallel for. Essa diferença de desempenho se manteve para os demais valores de MIN_BLK, sugerindo que a divisão mais flexível das tarefas em blocos menores nas chamadas recursivas com tasks permitiu um melhor aproveitamento dos recursos paralelos.

O valor de MIN_BLK influenciou significativamente o tempo de execução em ambas as abordagens. Conforme o valor de MIN_BLK diminuiu, o tempo de execução aumentou, tanto na abordagem parallel for quanto na abordagem com tasks. Isso ocorre porque, com valores menores de MIN_BLK, a quantidade de blocos aumenta, e a sobrecarga para gerenciar essas divisões também cresce. Para ambos os métodos, o menor valor de MIN_BLK (256) apresentou os piores tempos de execução, chegando a 13.1699 segundos com parallel for e 11.9433 segundos com tasks.

A variação entre execuções foi relativamente pequena em ambos os casos, mas a abordagem tasks apresentou uma leve vantagem em termos de consistência. Isso pode ser explicado pela forma como as tarefas são distribuídas dinamicamente com tasks, onde a criação e conclusão das tarefas é gerida de maneira mais eficiente em comparação ao modelo de parallel for, que depende da distribuição estática ou semi-estática das iterações entre as threads. Além disso, a abordagem parallel for tende a ter mais sobrecarga de sincronização, o que pode gerar flutuações dependendo da granularidade da divisão de tarefas.

Manipulação de Efeitos Colaterais no Vetor¶

A abordagem com pré-alocação de memória teve o melhor desempenho, com o tempo de execução médio (rodando 3 vezes) de 0.000286 segundos comparado com 0.001779 da abordagem utilizando o omp critical.

O uso de **#pragma omp critical** adicionou um overhead significativo. O overhead é justificado porque a diretiva critical força cada thread a esperar para acessar a região crítica onde o vetor é modificado (usando push_back()). Em outras palavras, mesmo com várias threads executando em paralelo, apenas uma thread por vez pode modificar o vetor, o que elimina o benefício do paralelismo para essa operação. Isso resulta em um tempo de execução muito maior do que a abordagem com pré-alocação, onde cada thread pode acessar e modificar sua posição no vetor de forma independente, sem necessidade de sincronização.

Além disso, a ordem dos dados não foi mantida na abordagem com ***#pragma omp critical***. As threads estão adicionando elementos fora de ordem no vetor, pois a operação **push_back()** não garante que os elementos sejam inseridos de acordo com o valor de i. Isso resulta em um vetor fora de ordem.

Por outro lado, a ordem dos dados foi mantida corretamente na abordagem com pré-alocação.Isso ocorre porque cada thread está escrevendo diretamente em uma posição fixa no vetor (definida por i), o que preserva a ordem dos dados de acordo com o valor de i.

Conclusão

Para o uso de schedulers no OpenMP, o scheduler auto demonstrou ser o mais eficiente em termos de tempo médio de execução, pois permitiu ao compilador escolher a melhor estratégia de distribuição. Já o scheduler default mostrou uma variação significativa, o que torna seu comportamento menos previsível.

No cálculo recursivo de Pi, a abordagem com tasks foi mais eficiente em comparação com parallel for, especialmente para blocos menores (MIN_BLK menores), devido à flexibilidade que as tarefas dinâmicas oferecem na divisão do trabalho. No entanto, à medida que o valor de MIN_BLK aumenta, o parallel for também melhora em termos de desempenho, mas com maior sobrecarga de sincronização.

Na manipulação de efeitos colaterais ao modificar vetores em paralelo, a pré-alocação de memória mostrou ser significativamente mais eficiente do que o uso de omp critical, que introduziu overhead devido à necessidade de sincronização entre as threads. Além disso, a ordem dos dados no vetor foi preservada apenas na abordagem com pré-alocação.

Para problemas recursivos, a abordagem com tasks mostrou-se mais eficiente, especialmente para distribuições dinâmicas de trabalho que precisam ser adaptadas com o tempo. Já para a manipulação de vetores com efeitos colaterais, a pré-alocação de

memória é a abordagem mais eficiente, pois elimina a necessidade de sincronização e mantém a ordem dos dados.