Relatório Atividade 13 - SuperComputação

Pedro Drumond

Schedules

1) Qual scheduler apresentou o menor tempo médio?

Os tempos de execução médios para cada scheduler variaram, mas os menores tempos foram observados com schedule(static), schedule(static, 8), e schedule(auto), com tempos em torno de 2.7x10^-6 segundos. Isso indica que os agendamentos estáticos com blocos maiores ou a configuração automática foram os mais rápidos.

2) Algum scheduler teve variações significativas entre as execuções? Se sim, por quê?

Sim, alguns schedulers apresentaram variações significativas:

Schedule(dynamic) e schedule(dynamic, 1): Esses agendamentos tiveram tempos mais altos (cerca de 5.188×10^−6.188 e 4.122×10^−6 segundos, respectivamente). Isso ocorre porque o agendamento dinâmico distribui as tarefas conforme as threads terminam seus trabalhos, o que pode resultar em sobrecarga de sincronização e tempos de execução variáveis.

3) Alguma característica específica do trabalho (como carga de dados, balanceamento) parece ter influenciado o comportamento de um scheduler em particular?

Sim, o comportamento foi influenciado pelo tamanho e pela distribuição das iterações:

Agendamentos estáticos (static): Esses agendamentos dividem as iterações igualmente entre as threads, o que funciona bem para cargas de trabalho uniformes e resulta em tempos de execução consistentes.

Agendamentos dinâmicos (dynamic): Foram menos eficientes devido à sobrecarga de distribuição dinâmica das iterações, o que causa variabilidade nos tempos de execução. A granularidade do agendamento (dynamic, 1 versus dynamic, 8) também influenciou o desempenho.

Agendamentos guiados (guided): Esses agendamentos começaram com blocos maiores e os diminuíram conforme o loop progrediu, equilibrando parcialmente a carga. Os tempos foram intermediários, mostrando que o balanceamento de carga pode ser melhorado dependendo da granularidade do agendamento.

Cálculo do PI

1) Qual abordagem (parallel for ou tasks) apresentou melhor desempenho?

Com base nos tempos registrados, a abordagem com parallel for mostrou tempos de execução mais consistentes e, em média, mais baixos. Os tempos variaram em torno de 1.33 a 1.38 segundos para os diferentes valores de MIN_BLK.

2) O valor de MIN_BLK ou o número de tarefas influenciou significativamente o tempo de execução?

Parallel for: Os resultados mostram que os diferentes valores de MIN_BLK não causaram grandes variações no tempo de execução, indicando que o tamanho do bloco não impactou significativamente o desempenho. Os tempos se mantiveram entre 1.33 e 1.38 segundos para os três valores testados.

Tasks: O número de tarefas teve um impacto maior nos tempos de execução. Com um número menor de tarefas (8 ou 16), houve uma variação significativa, com alguns tempos chegando a 2 segundos ou mais. À medida que o número de tarefas aumentou para 32 e 64, os tempos foram mais consistentes, mas ainda com algumas variações.

3) Alguma abordagem teve variação maior entre execuções? Por quê?

Sim, a abordagem com tasks apresentou uma variação maior. Isso ocorre porque a criação de tarefas dinâmicas pode introduzir sobrecarga adicional, especialmente quando o número de tarefas é baixo. Essa sobrecarga está relacionada à sincronização e ao gerenciamento de tarefas, o que pode impactar o desempenho de forma imprevisível, resultando em variações significativas entre as execuções.

Manipulação de Efeitos Colaterais no Vetor

1) Qual abordagem teve melhor desempenho: omp critical ou pré-alocação de memória?

A abordagem com pré-alocação de memória teve um desempenho melhor. Os tempos registrados para a pré-alocação variaram entre 0.00029 e 0.00030 segundos, enquanto os tempos com #pragma omp critical foram significativamente mais altos, variando entre 0.0014 e 0.0017 segundos.

2) O uso de omp critical adicionou muito overhead? Como você pode justificar isso?

Sim, o uso de #pragma omp critical adicionou overhead significativo. Isso ocorre porque a diretiva critical força as threads a se sincronizarem, garantindo que apenas

uma thread por vez possa acessar a seção crítica onde o vetor é modificado. Essa sincronização cria um gargalo, impedindo que as operações de inserção no vetor sejam realizadas em paralelo de forma eficiente.

Quando há muitas threads competindo para acessar a região crítica, o tempo de espera aumenta, o que pode anular os ganhos de paralelismo. Portanto, para operações que podem ser feitas sem sincronização explícita, como no caso da pré-alocação de memória, é preferível evitar critical.

3) A ordem dos dados no vetor foi mantida em ambas as abordagens?

Na abordagem com pragma omp critical, os elementos foram adicionados na ordem das iterações, garantindo que a ordem de inserção corresponda à ordem das operações sequenciais.

Na abordagem com pré-alocação de memória, cada índice do vetor foi preenchido diretamente pela thread correspondente, o que preserva a ordem das operações.

Conclusão

Abordagens para Paralelização Recursiva:

A paralelização utilizando parallel for mostrou ser mais eficiente e consistente em comparação com a abordagem de tasks. Os tempos de execução com parallel for foram mais baixos e tiveram menor variação, o que indica que essa abordagem é mais adequada para tarefas paralelizáveis de forma uniforme.

A abordagem com tasks apresentou maior variabilidade nos tempos, especialmente para um número menor de tarefas. Isso ocorre devido à sobrecarga associada à criação e sincronização das tarefas, que pode anular os benefícios do paralelismo em cenários onde o número de tarefas é insuficiente para dividir a carga de trabalho de forma eficiente.

Manipulação de Efeitos Colaterais:

A pré-alocação de memória foi muito mais eficiente do que o uso de #pragma omp critical. A sincronização com critical introduziu um overhead significativo, limitando o paralelismo e aumentando os tempos de execução.

A abordagem de pré-alocação eliminou a necessidade de sincronização explícita, permitindo que cada thread escrevesse em seu próprio índice do vetor de forma independente, resultando em um desempenho significativamente melhor.

Qual Abordagem Geral é Mais Eficiente?

Para problemas recursivos e com efeitos colaterais, a combinação de pré-alocação de memória e o uso de parallel for é a abordagem mais eficiente. Essa combinação evita a necessidade de sincronização explícita e divide o trabalho de forma balanceada entre as threads, resultando em tempos de execução mais rápidos e consistentes.

Resultados Inesperados

1. Variação Maior nos Tempos com tasks:

Os testes com tasks apresentaram variação maior do que o esperado. Isso pode ser explicado pela sobrecarga na criação de tarefas dinâmicas e pela sincronização envolvida, especialmente quando o número de tarefas era insuficiente para dividir o trabalho de forma eficiente.

Outro fator é que a granularidade das tarefas pode não ter sido ideal em algumas configurações, o que leva a uma distribuição desequilibrada do trabalho entre as threads.

2. Overhead Significativo com critical:

Embora esperado, o impacto do pragma omp critical foi ainda mais evidente. Esse resultado confirma que a sincronização explícita deve ser evitada sempre que possível em programas paralelos.