

# Provas Antigas - Comp. Paralela

Sophia  
Canezza



- 3) Explique o que significa variável shared e variável private no OpenMP. Considere aspectos práticos de execução de um código em uma memória compartilhada. No caso do Reduction, a variável entre parênteses, onde o operador atua, é shared ou private? Explique.

seu valor pode mudar  
entre os threads  
e nem se nota.  
"acessível"

Uma variável shared é a variável que é compartilhada entre as threads da região paralela, logo, se uma thread modificar seu valor, ela também estará modificada para outro processo paralelo. Já a variável private\*, é a qual seu valor é uma cópia da variável original, podendo ser manejada "localmente" por cada núcleo, sem a interferência de harm em outros. e exclusivo de cada thread

A operação Reduction do OpenMP cria uma cópia da variável para realizar as operações em cada núcleo paralelamente, sendo então uma variável private, para não ter de depender combinar os valores em uma operação final sobre uma variável shared comum.

\* sófe alterações apenas dentro de seu bloco de código  
↳ First Private = assume o valor da 1<sup>a</sup> vez que foi executada  
↳ Last Private = assume o valor da última vez que foi executada

- 4) Explique por que não é possível executar um programa paralelo com OpenMP em todos os nós de uma arquitetura de memória distribuída (e.g., cluster de computadores)?

Não é possível executar um programa com OpenMP em memória distribuída pois o OpenMP funciona com memória compartilhada, ou seja, as threads precisam estar presentes no mesmo espaço de memória. Um cluster de memória distribuída necessita de passagem de mensagens para os processos se comunicarem, algo que o OpenMP não apresenta.

e nem se nota.  
"acessível"

- 1) Explique: teste de escalabilidade forte e escalabilidade fraca para um programa paralelo em OpenMP em uma arquitetura com CPU de 64 núcleos.

Escalabilidade forte é quando o escalonamento dos threads

No teste de escalabilidade forte, mantemos o tamanho do problema fixo e aumentamos o n° de threads usadas (de 1 a 64), avaliando o quanto o tempo de execução diminui conforme aumentamos threads.

No teste de escalabilidade fraca, fazemos o número de threads e o tamanho do problema crescer proporcionalmente (nem caso, se 1 thread processa 1.000 itens, com 64 threads processará 16.000), para verificar se o tempo de execução permanece o mesmo.

2) Uma das técnicas do OpenMP para melhorar a escalabilidade de um código é o uso da cláusula schedule. Esta cláusula é baseada em três modos chamados de static, dynamic e guided. É possível estabelecer um tamanho de chunk, ou seja, quantidade de iterações, para escalonamento entre as threads. Assim, vários chunks são distribuídos entre as threads. O schedule é usado para melhorar o balanceamento de carga entre as threads objetivando a escalabilidade. Diferencie, os três modos listados apontando como eles podem ajudar a alcançar escalabilidade. Em resumo, só vantagens.

O modo static funciona retando previamente um tamanho de chunk e dividindo igualmente entre as threads esses chunks. Primeiro, ele é útil para loops平衡ados com pouca variabilidade, os quais sabemos que terão com média o mesmo tempo de execução durante a escalabilidade.

Ele gera pouca sobrecarga de sincronização, rende eficiente em loops de iteração semelhantes

Selecione o modo dynamic, por outro lado, associa cada chunk ao processo que está mais livre, seguindo uma política "first-come, first-served", sendo mais dinâmica e melhor para loops com cargas irregulares, aumentando a escalabilidade nestes casos.

Por último, o guided junta ambos os anteriores, expandindo e associando às threads os chunks em tamanhos grandes inicialmente, que diminuem exponencialmente até atingir um limite. Dessa forma, esse modo garante um balanceamento maior com menor overhead, já que no início com chunks grandes, há uma menor necessidade de comunicação e sincronização frequente entre threads. Ao

reduzir o tamanho dos chunk's, há um maior dinamismo entre as iterações de tamanhos variáveis. Assim, ele equilibra o baixo overhead inicial do static com a boa distribuição final do dynamic.

- 1) Explique: teste de escalabilidade forte e escalabilidade fraca para um programa em OpenMP, considerando uma arquitetura com CPU de 64 núcleos e memória compartilhada.

Lembrando em conta que a escalabilidade forte não aumenta o tamanho do problema e aumenta apenas o número de núcleos. É esperado uma diminuição no tempo proporcional ao aumento. Se 1 núcleo demora 64 segundos para finalizar, usando 64 núcleos o tempo deve ser de 1 segundo. Para a escalabilidade fraca o tamanho do problema aumenta proporcionalmente ao aumento de núcleos, sendo esperado um tempo constante. Então, se dobrar o problema e dobrar a quantidade de núcleos o tempo deve se manter igual.

outra  
resposta  
boa

Entre as comunicações coletivas do MPI, existem o Broadcast e o Scatter. Explique a diferença das duas.

ambas comunicações são One-to-All, porém, há uma principal diferença entre elas: o Broadcast envia a mesma mensagem para todos os demais processos, enquanto no scatter, o núcleo raiz possui um vetor com as mensagens divididas em partes, e ele envia cada parte a um processo.

Portanto, enquanto no broadcast todos os receptores recebem a mesma mensagem, no scatter eles recebem pedaços distintos delas.

- 3) Durante a execução de um código em OpenMP, o programador observou um ganho superlinear (ganho real > ganho ideal) após um teste de escalabilidade forte. De acordo com a Lei de Amdahl, isso não seria possível. No entanto, o experimento realizado está correto. Explique com base em uma alternativa, como foi possível o ganho superlinear.

Este gasto supérfluo pode ter ocorrido por causas externas às analisadas pela lei de Amdahl que o paralelismo trouxe indiretamente, como o uso + efetivo da memória cache, tendo + espaço para armazenar o volume de dados dos processos, reduzindo o tempo de acesso à memória RAM, ou como um melhor aproveitamento do tempo dos núcleos quando há um desbalanceamento de tempo de exec. de cada iteração, reduzindo gastos.

- 4) Explique se é possível ou não, executar um programa paralelo com OpenMP em todos os nós de uma arquitetura de memória compartilhada e distribuída (i.e., arquitetura NUMA).

OpenMP foi projetado para arquiteturas de memória compartilhada, funcionando em ambientes onde os threads podem aceder a uma área de memória. Porém, em ambientes de memória distribuída isso não é possível, pois ela necessita de mecanismos de passagem de mensagem entre memórias.

Em uma arquitetura híbrida como a NUMA, isso só seria possível se o OpenMP fizesse implementação separada nas áreas de memória compartilhada, e fizesse atráscado ao MPI para passagem de memória entre processos.