# SuperComp - Atividade 13

Giovana Cassoni Andrade.

## Schedules

Nessa atividade, o primeiro tópico são as schedules, sendo executado o código com os diferentes schedulers (static, dynamic, guided, auto) e registrando o tempo necessário para cada uma das 3 execuções feitas, bem como o tempo médio calculado, presentes na Tabela 1.

| | Execução 1 | Execução 2 | Execução 3 | Tempo médio |
|---|---|---|---|---|
| **static** | 2.57976e-06 s | 2.89083e-06 s | 3.0119e-06 s | 2.8274967e-06 s |
| **static 4** | 3.14787e-06 s | 2.66545e-06 s | 3.25963e-06 s | 3.0243167e-06 s |
| **static 8** | 5.29084e-06 s | 2.76882e-06 s | 3.01469e-06 s | 3.69145e-06 s |
| **dynamic** | 5.99492e-06 s | 4.96022e-06 s | 9.82825e-06 s | 6.9277967e-06 s |
| **dynamic 1** | 4.34276e-06 s | 3.58932e-06 s | 4.6622e-06 s | 4.1980934e-06 s |
| **dynamic 4** | 2.76137e-06 s | 2.74274e-06 s | 3.19444e-06 s | 2.8995167e-06 s |
| **dynamic 8** | 2.75485e-06 s | 2.82843e-06 s | 2.98303e-06 s | 2.8554367e-06 s |
| **guided** | 4.19375e-06 s | 3.93391e-06 s | 4.53368e-06 s | 4.2204465e-06 s |
| **guided 2** | 3.11248e-06 s | 2.81073e-06 s | 3.50084e-06 s | 3.14135e-06 s |
| **guided 4** | 2.84333e-06 s | 2.87499e-06 s | 3.2261e-06 s | 2.9814734e-06 s |
| **guided 8** | 2.91783e-06 s | 6.72508e-06 s | 2.89083e-06 s | 4.1779134e-06 s |
| **auto** | 2.70456e-06 s | 2.65986e-06 s | 2.6077e-06 s | 2.6573734e-06 s |

Tabela 1 - Tempo de execução dos 3 testes dos schedulers e o tempo médio.

Com todos os dados apresentados, afirma-se que o scheduler que apresentou o menor tempo médio foi o auto, com 2.6573734e-06 s, o que pode estar relacionado à sua natureza adaptativa e à capacidade de otimizar automaticamente a distribuição de carga com base nas condições de execução.

Em relação às variações nas execuções, pode-se apontar que o scheduler dynamic apresentou variações significativas, especialmente na configuração padrão (dynamic), com tempos entre 4.96022e-06 s e 9.82825e-06 s. Essa variação poderia ser causada pela natureza do scheduler dinâmico, que aloca pedaços de trabalho de forma imprevisível, dependendo da carga de trabalho de cada thread. Com pedaços pequenos ou um balanceamento entre as threads desigual, pode ocorrer variabilidade maior no tempo de execução. E ainda nesse ponto, o scheduler guided 8 também apresentou uma grande variação na segunda execução (6.72508e-06 s), o que pode ser consequência de como o guided distribui a carga: pedaços grandes no início e menores conforme o processamento avança, causando uma sobrecarga temporária em algumas threads.

Com essas análises feitas, afirma-se também que algumas características do trabalho, como pequena carga de dados e balanceamento uniforme, favorecem schedulers como static e auto, que conseguem otimizar a divisão de trabalho sem criar uma sobrecarga significativa.

## Cálculo do PI

Na segunda parte da atividade, são feitas paralizações no cálculo recursivo de Pi, usando parallel for e pragma omp task, respectivamente. Em ambos os códigos são realizadas mudanças em algum valor, MIN_BLK no programa parallel for e o número de tasks no programa task, e medidos os tempos de execução, todas as informações utilizadas e obtidas na Tabela 2.

| | **Execução 1** | **Execução 2** | **Execução 3** | **Execução 4** |
|---|---|---|---|---|
| **MIN_BLK** | **1024*1024*256** | **1024*1024*2** | **1024*128*2** | **512*2*2** |
| **parallel for** | 1.0780305 s | 1.07790852 s | 1.07818715 s | 1.07796435 s |
| **$n^0$ tasks** | **1 task** | **2 tasks** | **4 tasks** | **8 tasks** |
| **task** | 1.6189847 s | 1.03121962 s | 1.01196611 s | 1.08129677 s |

Tabela 2 - Tempo de execução dos 4 testes do cálculo pi com parallel for e task.

Com base nos tempos obtidos mostrados na Tabela 2, a abordagem de task apresentou melhor desempenho, especialmente com 4 tasks (1.0119 s), o que é inferior ao menor tempo do parallel for (1.0779 s).

Agora, abordando as mudanças feitas às execuções, o número de tarefas influenciou significativamente o tempo de execução na abordagem com tasks. Observa-se uma melhoria de desempenho à medida que o número de tasks aumenta, especialmente de 1 para 4 tasks. No caso do parallel for, a variação de MIN_BLK não parece ter causado grande diferença nos tempos.

Em relação às variações de tempo, a abordagem com tasks apresentou variação maior, principalmente nas execuções com 1 e 8 tasks (1.6189s e 1.0813s, respectivamente). Isso pode ser explicado pela forma como as tasks são distribuídas e agendadas dinamicamente, podendo gerar diferentes overheads dependendo do número de tasks e do trabalho a ser realizado em cada uma.

## Manipulação de Efeitos Colaterais no Vetor

A terceira e última parte de execuções e análises é referente à manipulação de efeitos colaterais no vetor, com a paralelização pragma omp critical para evitar acessos simultâneos ao vetor e uma modificação para pré-alocar a memória do vetor, registrando os tempos de execução dos 4 testes na Tabela 3.

| | **Execução 1** | **Execução 2** | **Execução 3** | **Execução 4** |
|---|---|---|---|---|
| **omp critical** | 2.67441e-04 s | 2.33723e-04 s | 2.81097e-04 s | 2.74085e-04 s |
| **pré-alocação** | 1.29098e-04 s | 1.15542e-04 s | 1.1121e-04 s | 5.24074e-05 s |

Tabela 3 - Tempo de execução dos 4 testes do código com omp critical e pré-alocação.

Analisando a tabela 3, afirma-se que a abordagem pré-alocação de memória apresentou o melhor desempenho, com os tempos menores que a metade do omp critical.

O uso de omp critical adicionou um overhead significativo, o que pode ser justificado pelo fato de que a diretiva impede que várias threads acessem o vetor simultaneamente, forçando-as a esperar umas pelas outras. Esse comportamento reduz a vantagem da paralelização, pois limita o paralelismo efetivo, o que impacta diretamente o tempo total.

Por último, em ambas as abordagens a ordem dos dados no vetor foi preservada. O uso de omp critical garante que as inserções ocorram de forma controlada, uma de cada vez.. Da mesma forma, na abordagem com pré-alocação de memória, como as inserções são feitas sem a necessidade de realocação dinâmica, a ordem dos dados também é mantida de forma previsível.

## Conclusões

Realizando uma conclusão das análises, o scheduler auto apresentou o melhor desempenho entre as opções testadas, graças à sua capacidade de balancear dinamicamente as threads, enquanto o dynamic e o guided mostraram maior variação nos tempos de execução. No cálculo recursivo de Pi, a abordagem com tasks superou o parallel for, especialmente com 4 tasks, embora tenha apresentado maior variação de tempos em algumas execuções, possivelmente devido ao overhead na distribuição das tasks. Na manipulação de efeitos colaterais no vetor, a pré-alocação de memória foi a abordagem mais eficiente, evitando o overhead significativo do omp critical, que limitou o paralelismo ao forçar sincronização entre as threads.

No geral, a abordagem com tasks mostrou-se mais eficiente para o cálculo recursivo de Pi, especialmente com um número de tasks adequado, e para lidar com efeitos colaterais em vetores, a pré-alocação de memória foi a solução mais eficiente.

Por fim, alguns resultados inesperados que podem ser levantados são o scheduler dynamic, que apresentou variações significativas nos tempos de execução, o que poderia ser explicado pela forma como as cargas de trabalho foram distribuídas de maneira imprevisível. Além disso, a abordagem com 8 tasks na paralelização de Pi não gerou o melhor tempo, algo que pode ser explicado pelo aumento no overhead de criação e gerenciamento de tasks quando seu número é excessivo.