UNIVERSIDADE FEDERAL DE PELOTAS

Relatório Paralelismo OpenMP

Guilherme Souza

Igor Barbosa

Thiago Heron

O presente relatório apresenta especificações de implementação, junto a resultados e observações sobre o trabalho optativo da cadeira , *Conceitos de Linguagem de programação*, ministrada pelo docente *Gerson Cavalheiro.*

O trabalho recebe como entrada **N** (tamanho do vetor), **M** (valor de 2 a 40) e um **P** (número de threads que formaram o time), trata se de um vetor de tamanho **N**, onde o valor do index da posição realizando a operação de **mod** com valor **M** (NmodM ou N%M), com o resultado obtido calcula-se o ***Fibonacci*** e armazena-o na posição.

Com isso implementou-se o trabalho fazendo uso da biblioteca *OpenMp* para realizar o paralelismo, utilizou-se:

- *#pragma omp parallel for schedule(static, chunk)*

Para realizar o paralelismo com um conjunto de posições o qual o for e encarregado, com a opção *static* e *chunk* definir um valor mais apropriado para uma possível otimização. Para realização do cálculo do Fibonacci, fez-se uso da estratégia *Task* o que apresenta um melhor desempenho e um encaixe com o cálculo executado, pois distribui-se as duas partes do cálculo entre as threads após finalizar é realizado um *Task Wait,* para que todos os resultado calculados em threads diferentes sejam unidos novamente e somados.

- *#pragma omp task shared(i)*
- *#pragma omp task shared(j)*
- *#pragma omp taskwait*

Para realizar uma melhor comparação e visualizar se tal ganho existia e se existente, até que ponto seria um ganho verdadeiro, implementou-se um código sequencial sem paralelismos, onde comparou-se o tempo com as mesmas entradas. Após alguma execuções observou-se alguns casos específicos como:

Para **N** extremamente pequenos, não se há ganho grandioso em paralelização, pois acaba por se tornar caro criar um time de threads para realizar uma operação tão curta, com isso o tempo de criação das *threads* junto ao processo de ciclo de vida dela, faz com o que código demore um pouco mais do que realmente deveria, isso se torna mais visível quando aumenta-se o número de integrantes do time de threads.

Outra situação na qual o ganho oscila, seguindo a mesma intenção de apresentada acima, quando se aumenta de mais o time de threads para um determinado **N**, para-se de ter um ganho (veja bem, tal observação feita em cima da máquina a qual foi submetida as execuções do código), ainda se tendo um ganho superior ao código sem paralelismo porém um tempo pior do que realizado com um time menor.

Apresentado alguns fatores que chamou atenção agora observou-se os reais ganhos ao paralelizar tal operação proposta. Com um **N** consideravelmente grande (veja que não definimos valor exato) o tempo de execução varia conforme o time de *threads* e chamado até o ponto de chegar na situação comentada acima, podendo-se ver de fato o tempo descendo enquanto o time de *threads* sobe e quando colocado ao lado do código sem paralelismo tal observação pode-se ver com mais clareza.

N	M	P	Real	User	Sys
1000000	8	2	*0m0.126s*	0m0.080s	0m0.008s
1000000	8	4	*0m0.093s*	0m0.072s	0m0.000s
1000000	8	6	0m0.086s	0m0.064s	0m0.000s
1000000	8	8	0m0.085s	0m0.064s	0m0.000s

Tabela 1: Execução do código que possui paralelismo, aumentando o número de *threads* de 2-8.

A tabela 1 demonstra os tempos de execução do código com um N considerável, onde é realizado o aumento das threads, podendo-se observar que ao iniciar o aumento do time, teve-se realmente um salto bem considerável, porém chegando a um extremo que seria 8 (novamente resultados obtidos de uma máquina específica), logo de 6 para 8 componentes do time não se teve um ganho verdadeiro.

Entretanto ao continuar aumentando o tamanho do N, chegamos a um ponto onde se observa-se

que o paralelismo deixa se ser eficiente, pelo menos em tempo ao equiparar ao código sem paralelismo, não demonstra um real ganho como pode ser visto na tabela 2 e 3.

N	M	P	Real	User	Sys
10000000	8	2	0m0.887s	0m0.676s	0m0.000s
10000000	8	4	0m0.808s	0m0.700s	0m0.012s
10000000	8	6	0m0.766s	0m0.656s	0m0.004s
10000000	8	8	**0m0.745s**	0m0.672s	0m0.008s

Tabela 2: Execução do código que possui paralelismo, aumentando o número de *threads* de 2-8.

N	M	P	Real	User	Sys
10000000	8	1	**0m0.490s**	0m0.392s	0m0.016s

Tabela 3: Execução do código sem paralelismo, o **P** e desconsiderado para cálculo.

Com tais resultado apresentados, observamos que o melhor tempo paralelizado, não foi melhor que o código que não há paralelização. Executando mais algumas vezes com um N de tamanho menor, observamos também que muitas vezes o código paralelo não consegue chegar a um tempo melhor que o não paralelo, onde seu melhor desempenho foi o mesmo valor em ambos ou com pouca diferença tendo uma melhora para o sem paralelização podendo ser visto na tabela 4, porém o paralelo sempre tendendo a alguns valores levemente maiores.

N	M	P	Real	User	Sys
1000000	8	1	0m0.080s	0m0.044s	0m0.000s
1000000	8	8	0m0.085s	0m0.064s	0m0.004s

Tabela 4: Comparação do tempo de execução do código não paralelizado (preenchido de cinza) e o paralelizado.

Partindo de tais resultado obtidos pode-se ver que que caímos algumas vezes na situação de que até que ponto vale a pena a paralelização, até que ponto vale a pena pagar pelo ciclo de vida das threads, pois traz um custo extra realizar suas chamadas, passar seus afazeres e depois dispensá las. Logo é importante ter isso em mente quando se pensar em paralelizar um código, se é viável ou não, se tal implementação permite e realmente há ganho com elas, dessa forma podendo chegar a um ponto máximo de otimização sempre uma perca ou trabalho desnecessario.