UNIVERSIDADE FEDERAL DE SÃO PAULO
INSTITUTO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA E TECNOLOGIA

Laboratório 02

Concorrência em memória compartilhada utilizando PThreads e OpenMP

Nomes: Ana Júlia de Oliveira Bellini **RA:** 111774
Willian Dihanster Gomes de Oliveira **RA:** 112269

SÃO JOSÉ DOS CAMPOS
2018

Introdução
Nesta atividade, o algoritmo para solução do problema N-Body é modificado, passando de serial a concorrente, utilizando PThreads e OpenMP. Com isso, são feitas diversas medições do desempenho do algoritmo, feitas estas duas modificações.

Metodologia
O algoritmo possui duas versões de implementação, ambas nas linguagens C, uma utilizando PThreads e outra utilizando OpenMP. Foram feitos testes com $N = 25000$ pontos, utilizando 1, 2, 4 ou 8 threads e posteriormente, foi feito o cálculo das medidas de desempenho, como o *speedup* e eficiência.

Experimentos
As especificações da máquina são descritas a seguir:

Processador: Intel (R) Core (TM) i5-7200U CPU @ 2.50GHz
Núcleos físicos: 2
Memória Cache: 3Mb L3
Memória RAM: 8 GB
Threads: 4
Hyperthreading: Sim
Sistema Operacional: Ubuntu 17.0 64 bits
Compilador: gcc 7.2.0

Resultados

- **Problema 1 - N-Body**

Número de Threads	Tempo execução em ms (PThreads)	Tempo execução em ms (OpenMP)
1	**9663**	**7798**
2	**9753**	**7833**
4	**9678**	**7827**
8	**9668**	**7841**

- **Problema 2 - Métricas**

Para este problema, considere que o número de threads é 2, utilizando a versão implementada em OpenMP.

a) A partir da versão serial do código fornecido, meça a fração (percentual) de tempo que o código executa apenas tarefas sequenciais, ou seja, tempo decorrido para o trecho do código que *não* é concorrente, e meça também a fração (percentual) de tempo que o código executa tarefas concorrentes.

Tempo total com o programa (TT) = 7836 ms

Tempo da seção paralela (TP) = 7833 ms

Tempo versão serial (TS) = (TT - TP) / TT = (7836 - 7833) = 3 ms

Fração de tempo de execução apenas de tarefas sequenciais: 3 / 7836 = 0.000382848

Fração de tempo de execução apenas de tarefas concorrentes: 7833 / 7836 = 0.999617

b) construa uma tabela com as estimativas de *Speedup* para execuções paralelas do código, a partir da fórmula da Lei de Amdahl, para 1, 2, 4, 8, 16, 32 e 64 processadores.

Considerando que a fórmula da estimativa de *Speedup* pela Lei de Amdahl é 1 / [s + (1 - s) / P], temos que:

Número de Processadores	Speedup
1	**1**
2	**1.99923**
4	**3.99541**
8	**7.97861**
16	**15.9086**
32	**31.6246**
64	**62.4927**

c) A partir das medidas de tempo em execuções paralelas efetuadas, construa outras duas colunas da tabela iniciada no item b, contendo o *Speedup* e eficiência paralela reais medidas nas execuções com 1, 2 e 4 *threads*.

No. de Proc.	Estim. de Speedup	Speedup	Eficiência Paralela
1	**1**	**1**	**1**
2	**1.99923**	**0.995531**	**0.497765**
4	**3.99541**	**0.996294**	**0.249073**

d) Compare os valores teóricos esperados de *Speed-up*, a partir da Lei de Amdahl (item b), com os valores reais medidos (item c). Analise os resultados.

Houve uma notável diferença de valores entre a estimativa e o resultado real. Para 1, 2 e 4 processadores, os Speed-ups ficaram próximos de 1, quase constantes. Enquanto isso, as estimativas foram praticamente lineares.

e) Modifique, a versão concorrente desenvolvida do código, alterando a função original: *double Random(void)* para que a mesma possa ser executada concorrentemente. Descreva também a principal limitação encontrada na versão original da função que gera números pseudo-aleatórios que impede que a mesma possa ser executada concorrentemente.

A principal dificuldade encontrada foi o fato da variável seed ser global e com isso, pode haver problemas com a concorrência, com mais de uma thread alterando a variável global ao mesmo tempo.

f) Refaça as medidas da tabela desenvolvida no item d com a nova versão do código. Implica em novas execuções paralelas da versão desenvolvida no item e.

No. de Proc.	Estim. de Speedup	Speedup	Eficiência Paralela
1	**1**	**1**	**1**
2	**1,999410476**	**1,013065064**	**0,506532532**
4	**3,996464943**	**0,9822650114**	**0,2455662529**

Conclusões

Pode-se concluir que ambas técnicas são eficiente na paralelização de aplicações. Mas, nesse caso, foi possível notar melhores desempenhos com o uso da biblioteca OpenMP, além da maior facilidade de implementação, o que são grandes vantagens em relação a Pthreads.
Para a análise com as medidas de desempenho, a parte serial do código se resume apenas a pequenas atribuições e códigos simples, logo, foram obtidos baixos valores para o valor *s,* que é o tempo gasto com parte seriais do código. E com isso, a estimativa de *Speedup* com a Lei de Amdahl não teve uma boa aproximação dos speedups reais. Também deve-se levar em consideração a arquitetura e especificações da máquina e do valor de *N* utilizado para tal comparação.
Com a alteração da função *random()* para ser paralelizada, o código foi quase todo paralelizado e com isso, houve ganhos no desempenho do algoritmo em relação a versão com a função *random()* serial, sendo assim, deve-se sempre que possível buscar e paralelizar regiões do código que possam ser paralelizadas.