Resolução – Exercício Programa 3 (Programação Paralela e Distribuída)

Fevereiro - 2023 / Leticia Bossatto Marchezi – 791003

Introdução

Semelhante ao EP2, este trabalho consiste no teste de desempenho da resolução da equação de transferência de calor de Laplace usando o método de Jacob e aplicando programação paralela. Entretanto, a biblitoeca a ser utilizada é a Open Multi-Processing(OpenMP).

O processo para execução dos testes pode ser descrito em 4 partes:

1) Código em C em que será realizada a solução da equação;

2) Arquivo makefile para compilação do código em C++;

3) Arquivo de definição do container Singularity;

4) Script para execução dos jobs no cluster HPC.

Resolução:

1. Código em C utilizando OpenMP
2. Makefile

```
CC=gcc

all: clean sequential openmp

sequential:
	$(CC) laplace_seq_iteracoes.c -o laplace_seq_it
openmp:
	$(CC) -fopenmp laplace_omp_iteracoes.c -o laplace_omp_it

clean:
	rm -f laplace_seq_it laplace_omp_it grid_laplace.txt grid_laplace_seq.txt
```

Figure 1: Makefile para compilação dos códigos em C

3. Definição do container

```
Bootstrap: docker
From: ufscar/ubuntu_mpich:latest

%help
	Paralelização da solução da equação de transferência de calor de Laplace pelo método de Jacobi usando OpenMP

%files
	./app/. /opt

%post
	echo "Compiling programs..."
	cd /opt && make
	cp laplace_omp_it laplace_seq_it /usr/bin/

%runscript
	exec $@
```

Figure 2: Arquivo de definição para build do container Singularity

4. Script do job

```
#!/bin/bash
#SBATCH -J laplace                    # Job name
#SBATCH -p fast                     # Job partition
#SBATCH -n 1                        # Number of processes
#SBATCH -t 01:30:00                 # Run time (hh:mm:ss)
#SBATCH --cpus-per-task=40          # Number of CPUs per process
#SBATCH --output=%x.%j.out          # Name of stdout output file
#SBATCH --error=%x.%j.err           # Name of stderr output file
lscpu

echo "*** SEQUENTIAL LAPLACE EQUATION GRID 1000X1000 ***"
srun singularity run container.sif laplace_seq_it 1000
echo " "
echo "*** OPENMP LAPLACE EQUATION grid 1000x1000 ***"

for j in 1 2 5 10 20 40
do
        export OMP_NUM_THREADS=$j

        echo "OPENMP PARALLEL LAPLACE EQUATION WITH $j THREADS"
        srun singularity run container.sif laplace_omp_it 1000
        echo "-----------------------------------------------"
        echo " "

done
```

Figure 3: Script a ser executado no cluster

A plataforma a ser utilizada para executar o job é o cluster HPC da UFSCar. O nó selecionado para executar o job possui as seguintes configurações:

```
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              96
On-line CPU(s) list: 0-95
Thread(s) per core:  2
Core(s) per socket:  24
Socket(s):           2
NUMA node(s):        8
Vendor ID:           AuthenticAMD
CPU family:          23
Model:               49
Model name:          AMD EPYC 7402 24-Core Processor
Stepping:            0
CPU MHz:             2794.661
BogoMIPS:            5589.32
Virtualization:      AMD-V
L1d cache:           32K
L1i cache:           32K
L2 cache:            512K
L3 cache:            16384K
NUMA node0 CPU(s):   0-5,48-53
NUMA node1 CPU(s):   6-11,54-59
NUMA node2 CPU(s):   12-17,60-65
NUMA node3 CPU(s):   18-23,66-71
NUMA node4 CPU(s):   24-29,72-77
NUMA node5 CPU(s):   30-35,78-83
NUMA node6 CPU(s):   36-41,84-89
NUMA node7 CPU(s):   42-47,90-95
```

Figure 4: Configuração do hardware do Cluster HPC

Execução

Resultados dos testes no cluster HPC da UFSCar

Resolução:

Após logar no servidor HPC, inserir os programas em c, os arquivos do container buildado e do job, o

job é submetido para execução na fila.

Os testes foram realizados com um grid de tamanho 1000x1000 para o algoritmo sequencial e paralelo, variando a quantidade de cores(threads) entre 1, 2, 5, 10, 20 e 40.

O output do job para o código sequencial é o seguinte:

```
*** SEQUENTIAL LAPLACE EQUATION GRID 1000X1000 ***
Jacobi relaxation calculation: 1000 x 1000 grid

Kernel executed in 26.412650 seconds with 3001 iterations
```

Figure 5: Resultado do código sequencial

E os tempos de execução para o algoritmo paralelizado com OpenMP:

```
*** OPENMP LAPLACE EQUATION grid 1000x1000 ***
OPENMP PARALLEL LAPLACE EQUATION WITH 1 THREADS

Kernel executed in 21.178272 seconds with 3001 iterations
-----------------------------------------------

OPENMP PARALLEL LAPLACE EQUATION WITH 2 THREADS

Kernel executed in 10.520368 seconds with 3001 iterations
-----------------------------------------------

OPENMP PARALLEL LAPLACE EQUATION WITH 5 THREADS

Kernel executed in 4.919117 seconds with 3001 iterations
-----------------------------------------------

OPENMP PARALLEL LAPLACE EQUATION WITH 10 THREADS

Kernel executed in 3.597055 seconds with 3001 iterations
-----------------------------------------------

OPENMP PARALLEL LAPLACE EQUATION WITH 20 THREADS

Kernel executed in 2.122642 seconds with 3001 iterations
-----------------------------------------------

OPENMP PARALLEL LAPLACE EQUATION WITH 40 THREADS

Kernel executed in 1.310838 seconds with 3001 iterations
-----------------------------------------------
```

Figure 6: Resultados do OMP

A partir do tempo sequencial e do tempo de execução paralelizado, é possível calcular a taxa de speed up com a seguinte fórmula:

$$Speedup = \frac{T_{sequential}}{T_{parallel}} \tag{2.1}$$

E a taxa de eficiência com a seguinte fórmula:

$$Eficiency = \frac{Speedup}{N_{Threads}} \tag{2.2}$$

Tabela de tempo de execução, speedup e eficiência para o algoritmo com PThreads

OpenMP table

	Threads	Execution Time	Speed Up	Efficiency
0	1	21.178272	1.247158	1.247158
1	2	10.520368	2.510620	1.255310
2	5	4.919117	5.369388	1.073878
3	10	3.597055	7.342854	0.734285
4	20	2.122642	12.443290	0.622165
5	40	1.310838	20.149439	0.503736

Figure 7: Tabela de speedup e eficiência

Gráfico de Speed Up pela quantidade de threads:

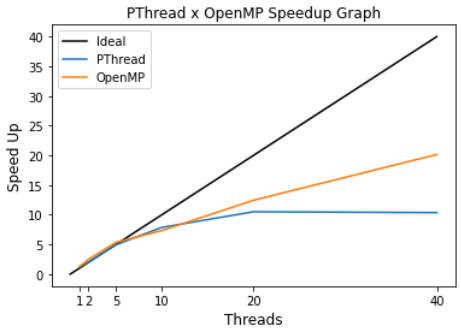

Figure 8: Gráfico PThread x OpenMP

Discussão dos resultados

Análise dos resultados

Resolução:

O resultado da paralelização do algoritmo com OpenMP possibilitou a diminuição do tempo de execução de 26,4s para 1,31s. Assim, o speed up no melhor caso é de 20(threads = 40), escalando progressivamente em conjunto com a quantidade de threads.

Diferente do código com PThreads, não há queda no speed up entre a execução de 20 threads para 40. Isso pode significar que a biblioteca OpenMP lida melhor com overheading de tarefas(muitos trabalhadores para poucas tarefas) e que o gasto de inicializar e gerenciar as threads é menor em comparação com PThreads.

Como o OpenMP possibilita a criação de uma região paralela e sincronização de threads entre os loops, há menor custo de desempenho em comparação com PThreads, que requere o uso de joins ou barreiras, que mesmo sendo mais proveitosas ainda são inferiores ao desempenho do OpenMP.

Dessa forma, é possível observar que a curva de speed up do OpenMP não se estabiliza para quantidades de threads acima de 20 e continua crescendo, tendo desempenho mais próximo ao ideal.

□