Resolução – Exercício Programa 4 (Programação Paralela e Distribuída)

Abril - 2023 / Leticia Bossatto Marchezi – 791003

Introdução

Este experimento tem como objetivo analisar a implementação do paralelismo de soma de vetores utilizando Message Passing Interface (MPI) na linguagem C.

O processo de experimentação teve 3 etapas:

1) Desenvolvimento do código em C utilizando MPI para solução da soma de vetores;

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

3) Testes de performance.

.

Resolução:

1. Código em C utilizando MPI

O paralelismo foi feito utilizando memória compartilhada, métodos de comunicação como Scatter e Gather e processos compartilhados.

2. Makefile

```
CC=gcc

all: clean mpi sequential openmp run

sequential:
    $(CC) vecadd_seq.c -o vecadd_seq

openmp:
    $(CC) vecadd_omp.c -o vecadd_omp -fopenmp

mpi:
    mpicc vecadd_mpi.c -O3 -o vecadd_mpi
run:
    mpirun -np 1 ./vecadd_mpi 9600000
    mpirun -np 2 ./vecadd_mpi 9600000
    mpirun -np 3 ./vecadd_mpi 9600000
    mpirun -np 4 ./vecadd_mpi 9600000
clean:
    rm -f vecadd_seq vecadd_omp vecadd_mpi
```

Figure 1: Makefile para compilação dos códigos em C

3. Testes de performance

A plataforma para execução dos testes foi o computador pessoal da aluna, que possuem as seguintes configurações:

```
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         43 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  8
  On-line CPU(s) list:   0-7
Vendor ID:               AuthenticAMD
  Model name:            AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
    CPU family:          23
    Model:               24
    Thread(s) per core:  2
    Core(s) per socket:  4
    Socket(s):           1
    Stepping:            1
    Frequency boost:     enabled
    CPU max MHz:         2100,0000
    CPU min MHz:         1400,0000
    BogoMIPS:            4199.70
```

Figure 2: Configuração do hardware do laptop

Arquitetura: x86_64

Processador: AMD Ryzen 5 3500U

Threads por core: 2

Cores por socket: 4

Sockets: 1

Os resultados dos testes de performance serão abordados na próxima seção.

Execução

Resultados dos testes de desempenho

Resolução:

O resultado da execução do código sequencial com array de 9600000 elementos pode ser visualizado a seguir.

```
vectors added with 0 errors in 0.187502 seconds
```

Figure 3: Resultado do código sequencial

Os testes de performance foram realizados utilizando arrays de 9600000 elementos e com variação entre 1 a 4 processos.

A execução do código paralelizado foi feito a partir do próprio makefile.

```
mpirun -np 1 ./vecadd_mpi 9600000
Soma completa com 0 erros em 0.033960 segundos usando 1 processo(s)

mpirun -np 2 ./vecadd_mpi 9600000
Soma completa com 0 erros em 0.025145 segundos usando 2 processo(s)

mpirun -np 3 ./vecadd_mpi 9600000
Soma completa com 0 erros em 0.021227 segundos usando 3 processo(s)

mpirun -np 4 ./vecadd_mpi 9600000
Soma completa com 0 erros em 0.020006 segundos usando 4 processo(s)
```

Figure 4: 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 de MPI

MPI Table

	Core	Execution Time	Speed Up	Efficiency
0	1	0.030323	6.183491	6.183491
1	2	0.024893	7.532318	3.766159
2	3	0.021368	8.774897	2.924966
3	4	0.020895	8.973534	2.243384

Figure 5: Tabela de speedup e eficiência

Gráfico de Speed Up pela quantidade de cores:

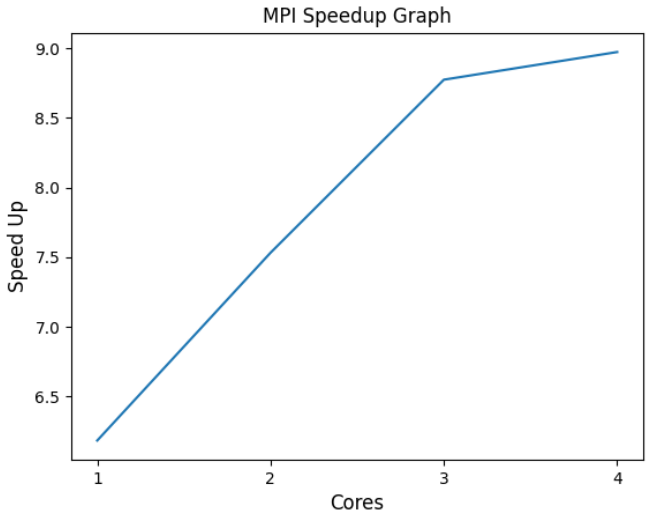

Figure 6: Gráfico MPI

Discussão dos resultados

Análise dos resultados

Resolução:

O resultado da paralelização pode ser indecisivo pois há pouca amostragem para análise, já que há apenas 4 cores na máquina de teste. Para 1, 2 e 3 cores o crescimento da curva do speedup foi linear e próximo ao ideal, entretanto houve uma queda a partir do uso de 4 threads, que pode ser explicado pelo overhead do gerenciamento de cores para pouco trabalho a ser feito.

□