Relatório 9

Programação de Alto Desempenho (HPC) usando Open MP –

Computação Paralela

Cristiano Lopes Moreira

Matrícula: 119103-0

Sumário

1. Introdução

A crescente demanda por sistemas de uso em tempo real em conjunto com os limites físicos que delimitam o crescimento da capacidade de processamento dos computadores vem desde 1955, pela iniciativa da IBM com o arquiteto de computadores Gene Amdahl, impulsionando desenvolvimento de tecnologias de Programação de Alto Desempenho (HPC) e possibilitando a interconexão de múltiplos computadores e processadores com suas unidades de memórias para dividir uma tarefa e compartilhar a capacidade de cada unidade de processamento.

Dentre as técnicas HPC desenvolvidas encontra-se a técnica de computação paralela via open Multi Processing que compartilha a memória de 01 processo entre múltiplas threads executadas entre multi processadores, com métodos consolidar as variáveis que são tratadas individualmente pelas threads.

O objetivo deste trabalho é implementar a tecnologia de computação paralela utilizando open MP, verificar e quantificar seus ganhos e perdas na tarefa de cálculo numérico de integral pelo método de Newton-Cotes pontos médios.

2. Desenvolvimento teórico

Como já afirmado por Amdahl (1967), a décadas os profetas expressaram a alegação de que a organização de um único computador atingiu seus limites, e que avanços verdadeiramente significativos podem ser feitos apenas pela interconexão de uma multiplicidade de computadores de maneira a permitir uma solução cooperativa.

Geralmente, métodos de computação paralela têm sido usados para aumentar o desempenho do sistema ou aumentar sua disponibilidade, porém a busca por esta interconexão trilha os desafios de conflitos no uso de recursos simultâneos como memória, discos, i/o. Três pontos são limitantes no progresso da computação com

multiprocessadores: custo, facilidade de aplicação e desempenho (RODGERS, 1985).

No que tange a custos, apartado do elevado custo de memória cache (tipicamente integrada direto na CPU), que reduz os problemas de conflitos e colisões no uso de memória entre os processadores; o crescimento da oferta dos elementos de hardware nos anos entre 2017 e 2019 reduziu os valores dos elementos computacionais que não se mostra mais um grande limitante para a computação com multiprocessadores.

No limitante de facilidade de aplicação é possível verificar que as práticas de design de sistemas de hardware e software que produzem um bom desempenho de processador monolítico quando aplicadas a multiprocessadores levam a gargalos de desempenho devido à largura de banda de memória e falta de recursos de serviços do sistema. O próprio sistema operacional gera algumas dependências de serialização, o que limita o desempenho da adição de múltiplos processadores, sendo o ponto mais importante a interação física do sistema de interconexão com o sistema de memória, já que os processadores compartilham estes recursos.

Uma das técnicas utilizadas para contornar o desafio do compartilhamento dos recursos de memória é a arquitetura de endereçamento virtual de memória, mas além de adicionar uma complexidade na gestão de tabelas de indexação de memória, cria uma latência adicional e consequente Overhead de gerenciamento do paralelismo (podendo variar de acordo com o tamanho do problema e número de núcleos utilizados) que culminando em uma redução do desempenho da aplicação [3] e [4].

Outra abordagem, ainda na parte de aplicação, é a definição de arquiteturas para uso da memória e do gerenciamento das aplicações:

Estrutura de conexão de memória por

- Multiprocessadores com barramento compartilhado no tempo
- Multiprocessadores com chave de barra cruzada

- Multiprocessadores com memória multiportas

Gestão das aplicações por

- Supervisor separado em cada processador
- Sistema operacional mestre-escravo
- Sistema operacional de supervisor flutuante

De todas as formas os sistemas de multiprocessadores compartilham (memoria, I/O, devices, interrupções de sistemas etc.) e recursos como memória e I/O são atribuídos dinamicamente a processos, e não permanentemente conectados aos processadores (RODGERS, 1985).

Nos estudos realizados por Amdahl (1967) ele já observava que a fração computacional responsável por gerenciamento de memória corresponde cerca de 40%; esse fato somado as instruções que são necessariamente sequencias, como I/O, geram um gargalo a todo o processamento e coloca um limite superior na aceleração

Aceleração $$S = \frac{1}{r_s + \frac{r_p}{n}} \quad (1)$$

Onde $r_s+r_p=1$ (2) e r_s representa a proporção da porção sequencial em um programa.

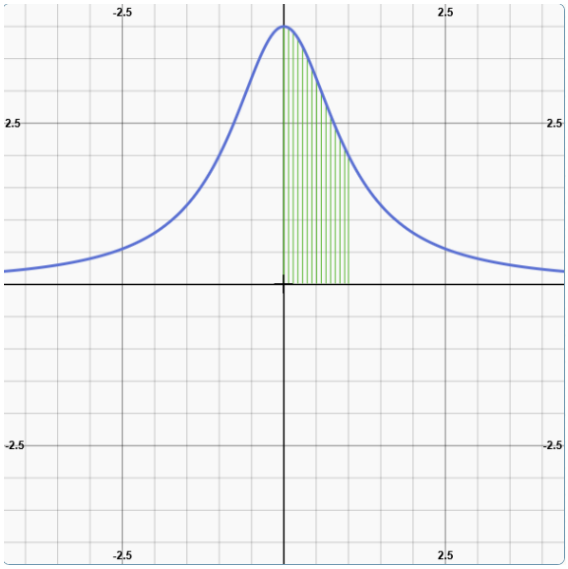
Levando (2) em (1) $$S = \frac{1}{(1 - r_p) + \frac{r_p}{n}} \quad (3)$$

2.1. Descrição do problema:

Deseja-se calcular a integral da função abaixo utilizando o método dos Pontos Médios, com adaptação do resultado pela variação da taxa de erro, implementada

com computação paralela utilizando openMP, e avaliar a eficiência do paralelismo em processadores multicores de 8 cores.

Função $$y = \frac{4}{1 + x^2} \quad (4)$$



2.2. Algoritmo de Newton, método dos pontos médios, com OpenMP:

O algoritmo de integração numérica pelo método dos pontos médios, de integração por aproximação, utiliza a metodologia de Newton-Cotes variando as fórmulas de Newton-Cotes em cada caso, para calcular a função integral determinada. Suas operações consistem em calcular o a função f(x), calcular a integral da função f(x), calcular a precisão da integral, calcular e retornar o número de subintervalos são necessários para uma dada precisão.

Sua integração utilizando openMP é realizada pela divisão da tarefa no loop “for” entre “n” threads, de um mesmo processo, que compartilham a memória em suas interações e agrupa o resultado de cada thread pela rotina “reduction” do openMP.

Seus métodos principais são:

Seus métodos principais são:

construtor (): inicializa a função.

rg_medio (x0, x1, erro): recebe o intervalo superior, intervalo inferior e taxa de erro aceitável, e realiza os cálculos e divisões dos intervalos para retornar o valor da integral utilizando a regra dos pontos médios.

rg_trapezio (x0, x1, erro): recebe o intervalo superior, intervalo inferior e taxa de erro aceitável, e realiza os cálculos e divisões dos intervalos para retornar o valor da integral utilizando a regra trapezoidal.

rg_simpson (x0, x1, erro): recebe o intervalo superior, intervalo inferior e taxa de erro aceitável, e realiza os cálculos e divisões dos intervalos para retornar o valor da integral utilizando a regra de simpson.

prtAccurate(), retorna a taxa de erro no cálculo da derivada;

qtPassos, retorna o número de intervalos necessários para calcular a integral com a taxa de erro aceitável.

opm_parallel for: divide o processo entre as threads e pelo reduction agrupa as váriáveis

3. Proposta de implementação

Pseudocódigo:

Adiciona biblioteca omp

Calcula integral (double a, double b, double erro)
    delta = (a+b)/2
    Integral = calcula integral (a, b);
    Erro = calcula erro integral
    enquanto (erro > precisao)

```
        delta = delta /2
        integral=0
        erro=0
#inicializa open MP no loop com redução para integral e erro
        para passo = início até passo < fim, incrementa passo +1
                x= (passo*delta+ passo*delta+delta)/2
                integral = integral + delta * calculaf(x);
                erro = erro +calcula erro integral
Retorna integral
```

4. Experimentação e Resultados

Foram executados 230 ciclos com 16.7 milhões subdivisões sendo distribuída a tarefa entre threads de um mesmo processo utilizando openMP da seguinte forma: 1 acrescido em 1 até 10 threads, de 10 acrescido de 10 até 100 threads e de 100 acrescido de 50 até 250; como observação inicial do resultado comparado com a aplicação executada sem nenhuma técnica de paralelismo.

Ambiente:

Processoador model name: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 8 cores

OS Linux openSUSE Leap 15.1

Resultado do cálculo da área da sob a função f(x)=4/(1+x^2)

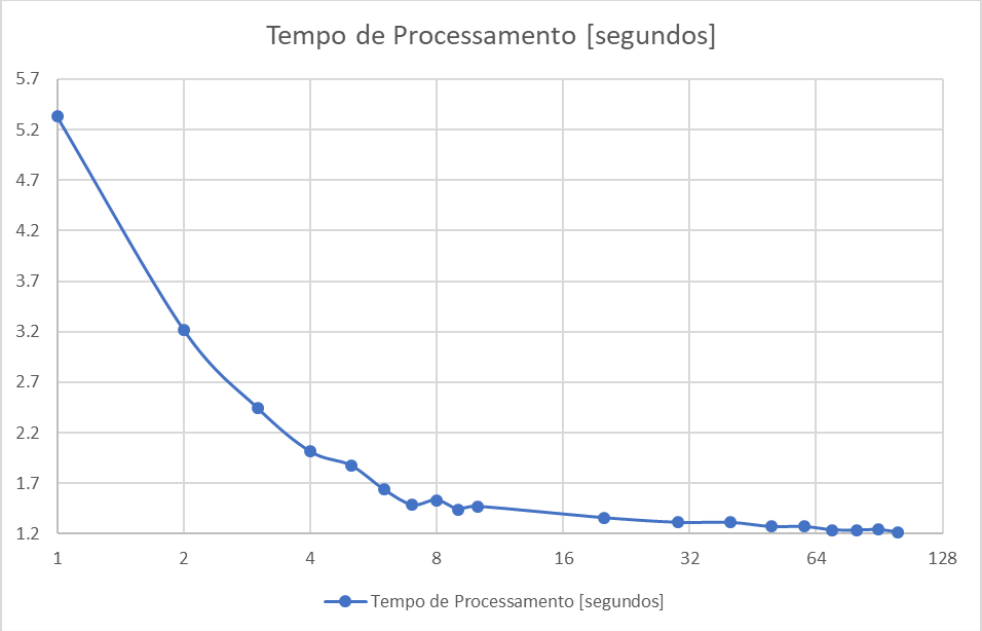
Newton-Cotes regra dos pontos médios		
N	Integral	Erro
16.777.215	3.1415924152	-0.00000000000000473695

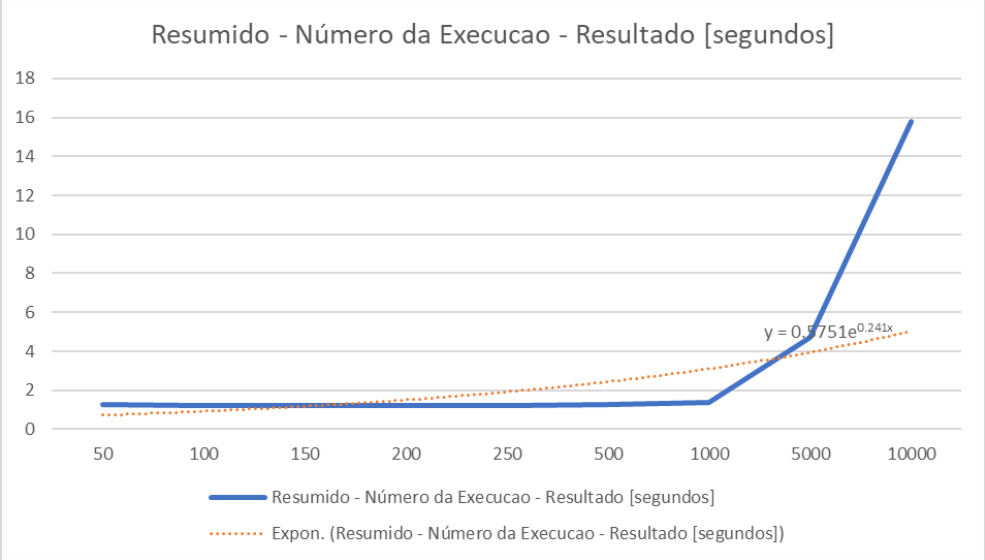
Tempo de execução por threads com openMP

	Número da Execução - Resultado [segundos]										
threads	1	2	3	4	5	6	7	8	9	10	Média
S/openMP	4.72	4.74	4.7	4.73	4.7	4.74	4.63	4.7	4.96	4.66	**4.728**
1	7.94	5.22	5.45	5.06	4.69	4.89	5.04	4.96	5.04	5	**5.329**
2	3.01	3.26	3.3	3.07	3.43	3.11	3.32	3.29	3.31	3.09	**3.219**
3	2.28	2.34	2.4	2.39	2.65	2.38	2.44	2.57	2.44	2.54	**2.443**
4	1.85	1.89	1.91	2.02	2.16	2.01	2.06	2.17	2.05	2.02	**2.014**
5	1.8	2.02	1.8	1.96	2.02	1.82	1.9	1.68	1.94	1.81	**1.875**
6	1.66	1.55	1.61	1.72	1.77	1.73	1.56	1.55	1.69	1.52	**1.636**
7	1.4	1.6	1.53	1.48	1.46	1.42	1.46	1.49	1.46	1.55	**1.485**
8	1.55	1.58	1.56	1.6	1.49	1.5	1.51	1.51	1.5	1.53	**1.533**
9	1.44	1.41	1.43	1.42	1.46	1.44	1.45	1.44	1.48	1.44	**1.441**
10	1.37	1.42	1.49	1.66	1.57	1.43	1.42	1.44	1.46	1.43	**1.469**
20	1.39	1.34	1.36	1.35	1.39	1.34	1.36	1.35	1.35	1.34	**1.357**
30	1.32	1.25	1.29	1.32	1.31	1.35	1.33	1.32	1.32	1.3	**1.311**
40	1.36	1.33	1.34	1.3	1.26	1.27	1.33	1.31	1.29	1.37	**1.31**
50	1.27	1.26	1.3	1.24	1.25	1.27	1.3	1.27	1.27	1.26	**1.269**
60	1.28	1.24	1.39	1.24	1.23	1.29	1.25	1.28	1.24	1.25	**1.269**
70	1.24	1.22	1.22	1.23	1.21	1.23	1.27	1.24	1.24	1.26	**1.236**
80	1.22	1.24	1.26	1.25	1.24	1.23	1.22	1.23	1.22	1.22	**1.233**
90	1.24	1.23	1.24	1.24	1.25	1.24	1.24	1.23	1.24	1.24	**1.239**
100	1.21	1.19	1.22	1.2	1.21	1.24	1.22	1.21	1.23	1.22	**1.215**
150	1.18	1.18	1.22	1.21	1.18	1.2	1.22	1.22	1.22	1.2	**1.203**
200	1.21	1.22	1.21	1.23	1.23	1.22	1.26	1.21	1.2	1.19	**1.218**
250	1.2	1.22	1.22	1.21	1.22	1.19	1.2	1.24	1.22	1.21	**1.213**

Tempo de execução por threads com openMP

	Número da Execução - Resultado [segundos]										
threads	1	2	3	4	5	6	7	8	9	10	Média
50	1.27	1.26	1.3	1.24	1.25	1.27	1.3	1.27	1.27	1.26	**1.269**
100	1.21	1.19	1.22	1.2	1.21	1.24	1.22	1.21	1.23	1.22	**1.215**
150	1.18	1.18	1.22	1.21	1.18	1.2	1.22	1.22	1.22	1.2	**1.203**
200	1.21	1.22	1.21	1.23	1.23	1.22	1.26	1.21	1.2	1.19	**1.218**
250	1.2	1.22	1.22	1.21	1.22	1.19	1.2	1.24	1.22	1.21	**1.213**
500	1.28	1.26	1.26	1.23	1.26	1.24	1.24	1.25	1.24	1.24	**1.25**
1000	1.38	1.41	1.39	1.38	1.38	1.37	1.37	1.39	1.42	1.38	**1.387**
5000	4.74	4.64	4.53	4.57	4.67	4.77	4.81	4.96	4.76	4.71	**4.716**
10000	15.37	16.92	15.66	15.99	15.63	15.53	16.28	15.26	15.6	15.6	**15.784**


Tempo de Processamento [segundos]
5.7
5.2
4.7
4.2
3.7
3.2
2.7
2.2
1.7
1.2
1
2
4
8
16
32
64
128
Tempo de Processamento [segundos]


Resumido - Número da Execucao - Resultado [segundos]
18
16
14
12
10
8
6
4
2
0
50
100
150
200
250
500
1000
5000
10000
y = 0.5751e0.241x
Resumido - Número da Execucao - Resultado [segundos]
Expon. (Resumido - Número da Execucao - Resultado [segundos])

5. Conclusão

O acréscimo de threads paralelos até o máximo do número de cores do processador acelera a execução da tarefa computacional, porém não proporcional à quantidade de cores adicionadas, é observada reduções do acréscimo de aceleração a cada processador adicionado em confirmação da lei de Amdahl.

Para acréscimos de threads acima do número de cores do processador observa-se que o comportamento é contrário, desacelerando a execução da tarefa computacional de forma exponencial com a quantidade de processos adicionados, o que confirma os custos de Overhead mensurados nos trabalhos de Höfinger e Haunschmid (2017) e Oliveira et al. (2018).

6. Referências

[1] AMDAHL, Gene M.. Validity of the single processor approach to achieving large scale computing capabilities. In: AFIPS SPRING JOINT COMPUTER CONF, 67., 1967, Atlantic City. **(Spring) Proceedings of the April 18-20**. Reston: Afips Press, 1967. v. 30, p. 483 - 485

[2] RODGERS, David P.. Improvements in Multiprocessor System Design. In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 12., 1985, Boston. **ISCA '85 Proceedings**. Los Alamitos: Ieee Computer Society Press, 1985. p. 225 - 231.

[3] HÖFINGER, Siegfried; HAUNSCHMID, Ernst. **Modelling parallel overhead from simple run-time records**. 2017. The Journal of Supercomputing. Disponível em: <https://doi.org/10.1007/s11227-017-2023-9>. Acesso em: 31 mar. 2017.

[4] OLIVEIRA, Victor H. F. et al. Application Speedup Characterization: Modeling Parallelization Overhead and Variations of Problem Size and Number of Cores. In: ACM/SPEC INTERNATIONAL CONFERENCE ON PERFORMANCE ENGINEERING, 18., 2018, Berlin. **Proceeding ICPE '18 Companion of the 2018**. New York: Acm, 2018. p. 43 - 44