CENTRO UNIVERSITÁRIO FEI

VICTOR BIAZON
RA: 119.115-4

RELATÓRIO VIII – PROGRAMAÇÃO CIENTIFICA
INTEGRAL DE NEWTON COM OPENMP

SÃO BERNARDO DO CAMPO
2019

VICTOR BIAZON
RA: 119.115-4

RELATÓRIO VIII – PROGRAMAÇÃO CIENTIFICA
INTEGRAL DE NEWTON COM OPENMP

Relatório de desenvolvimento programação paralela para executar o algoritmo de integral de Newton, desenvolvido pelo aluno Victor Biazon, RA 119.115-4, para disciplina PEL216 – Programação Cientifica, ministrada pelo professor Reinaldo Bianchi.

São Bernardo do Campo
2019

Sumário:

Motivação

Implementação de programação paralela para melhorar desempenho do algoritmo de integral pelo método de Newton.

Objetivo:

Desenvolver algoritmo para execução do algoritmo de integral pelo método de Newton utilizando de programação paralela para que seja possível diminuir o tempo necessário para execução do mesmo. Verificar o uso do OpenMP no Linux como forma de utilizar a programação paralela.

Teoria

OpenMP

“OpenMP is an Application Program Interface (API), jointly defined by a group of major computer hardware and software vendors. OpenMP provides a portable, scalable model for developers of shared memory parallel applications. The API supports C/C++ and Fortran on a wide variety of architectures.” (BLAISE, 2019).

“By itself, OpenMP parallelism is limited to a single node; For High Performance Computing (HPC) applications, OpenMP is combined with MPI for the distributed memory parallelism. This is often referred to as **Hybrid Parallel Programming**.

OpenMP is used for computationally intensive work on each node

MPI is used to accomplish communications and data sharing between nodes” (BLAISE, 2019).

O API OpenMP como descrito por Blaise (2019) em seu site mantido pelo LLNL, é utilizado principalmente para paralelizar processos em c++ e Fortran em computadores de memória compartilhada, ou seja, apenas utilizando os cores de um único computador. Assim, se necessário mais poder de processamento, é necessário associa-lo ao MPI para que se utilizem vários PC`s conectados entre si, executando as tarefas e compartilhando as informações para chegar ao resultado. Esta ultima arquitetura é chamada de Programação Paralela Hibrida, da sigla HPP em inglês.

Implementação

Para a implementação do OpenMP foi utilizado o mesmo código anteriormente desenvolvido para integral pelo método de Newton em C++ com a adesão da biblioteca e dos comandos específicos do API conforme abaixo. Os comandos específicos estão em negrito.

```
O código abaixo foi desenvolvido em C++ e compilado no Linux Ubuntu 18.04 com a
biblioteca MPIC++ com os comandos:

Compilar:     g++ -fopenmp <nome do aquivo.extensao> -o <nome do arquivo.executavel>
Executar:     . /<nome do arquivo.executavel>
```

Código:

Foram descritos abaixo apenas as partes relacionadas com o OMP.

```
#include <omp.h>
    //configura o OMP
    int nprocs = omp_get_num_procs();
    omp_set_num_threads(nprocs);
float IntegralNumericaIterativaOMP (float a, float b, float erro){
    int n = 2048; //inicializa variaveis
    float intervalo = 0;
    float totalIntegral = 0;
    float totalerro = 0;
    int contador = 0;

    do {
        intervalo = (a + b)/n; //separa integração em intervalos
        totalerro = 0;
        totalIntegral = 0;

        #pragma omp parallel for shared(contador) private(tid) //informa ao compilador
que o for é paralelo
        for (contador = 0; contador < n; contador++){
        float a1 = contador * intervalo;
        float b1 = (contador + 1) * intervalo;

        totalIntegral += Simpson(a1, b1); //calcula a aproximacao pelo metodo de
Simpson
        totalerro += SimpsonErro(a1, b1); //calcula o erro da aproximacao

        }

    }
    while (false);//(abs(totalerro) > erro);
    cout << "Integral OpenMP: " << totalIntegral << " erro: " << totalerro << " numero
de divisoes: " << n << endl;
}
```

Experimentos e resultados

Para testar o algoritmo foi proposta uma comparação entre a iteração com um processador apenas, e com quatro processadores:

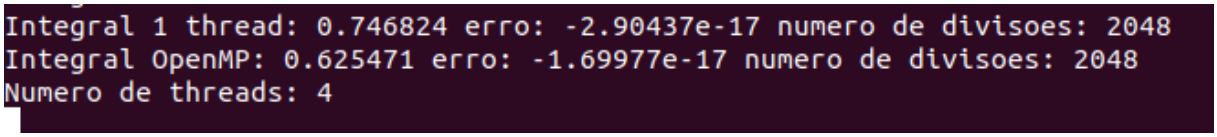

Fig. 1 Comparação entre tasks.

Como visto no experimento o OpenMP, da forma que foi implementado retorno um valor diferente do esperado, provavelmente devido ao Racing entre os cores no acesso a memória onde estava sendo acumulado o valor da integral. No entanto a execução foi cerca de três vezes mais rápida com o OMP, sendo para um core 145ms e para quatro cores 43ms. Não sendo exatamente quatro vezes mais rápido devido ao curto do FORK/JOIN do OMP.

Para tarefas maiores o OMP mostra que a performance é inversamente proporcional ao número de cores, desde que não haja FORKs/JOINs utilizados de maneira equivocada.

Trabalhos correlatos

Distributed TensorFlow with MPI

Abhinav Vishnu Charles Siegel Jeffrey Daily

https://www.researchgate.net/publication/301842011_Distributed_TensorFlow_with_MPI

Efficient Large Message Broadcast using NCCL and CUDA-Aware MPI for Deep Learning ∗

A. A. Awan K. Hamidouche A. Venkatesh D. K. Panda

https://dl.acm.org/citation.cfm?id=2966912

Conclusão

Como pode-se notar o processamento paralelo diminui muito o tempo necessário para desempenhar uma tarefa quando a mesma é repetitiva e independente do resultado de outras iterações. Portanto para o método de Newton e outros que seu resultado é não dependente de resultados anteriores, o processamento paralelo é uma ferramenta importante para diminuir o tempo necessário para chegar ao resultado sem perder a qualidade do mesmo. No entanto, é necessário considerar o tempo de execução do FORK e JOIN na execução do programa para evitar que se aumente o tempo desnecessariamente.

Referências bibliográficas

[1] BLAISE, Barney. **OpenMP Tutorial.** Disponível em: < https://computing.llnl.gov/tutorials/openMP/> Acesso realizado: 09 Set 2019.