CÁLCULO ESTIMATIVO DA ÁREA DE UM CONJUNTO DE MANDELBROT USANDO OPENMP[1]

Marcelo Bernardy de Azevedo[2] <marcelo.bernardy@acad.pucrs.br>
Thomas Pozzer Moraes[3] <thomas.pozzer@acad.pucrs.br>
Roland Teodorowitsch[4] <roland.teodorowitsch@pucrs.br> – Orientador

Pontifícia Universidade Católica do Rio Grande do Sul – Faculdade de Informática – Curso de Engenharia de Software
Av. Ipiranga, 6681 Prédio 32 Sala 505 – Bairro Partenon – CEP 90619-900 – Porto Alegre – RS

21 de Maio de 2020

RESUMO

Este artigo descreve uma solução para o primeiro trabalho proposto na disciplina de Programação Paralela do curso de Engenharia de Software na PUCRS, que aborda a paralelização da solução da Área de um Conjunto de Mandelbrot usando OpenMP, incluindo a avaliação de desempenho do algoritmo e contextualização sobre a utilização do processamento paralelo para obtenção de um melhor desempenho.

Palavras-chave: OpenMP; Mandelbrot; Programação Paralela;

ABSTRACT

Title: *"Area of a Mandelbrot set using OpenMP"*

This article describes a solution for the first work proposed in the Parallel Programming discipline of the Software Engineering course at PUCRS, which approach the parallelization of the Mandelbrot Set Area solution using OpenMP, including the performance evaluation of the algorithm and contextualization on the use of parallel processing to obtain better performance.

Key-words: *OpenMP; Mandelbrot; Parallel Programming;*

1 INTRODUÇÃO

O problema descrito na proposta do trabalho é de calcular a área de um conjunto de Mandelbrot. O Conjunto de Mandelbrot é um conjunto de números complexos *c* para os quais, a partir de uma condição inicial *z=c*, a iteração de *z=z2+c* não diverge. Para determinar se um ponto *c* está neste conjunto, deve-se realizar determinado número de iterações sobre z, sendo que, se ao final dessas iterações tivermos $|z| > 2$, então o ponto está fora do conjunto de Mandelbrot.

Neste artigo vamos mostrar a experiência de paralelizar uma versão sequencial de um código que realiza um cálculo estimativo da Área de um Conjunto de Mandelbrot (BULL, 2019). Nesse código, geram-se pontos de forma homogênea sobre uma área quadrada e para cada ponto realiza-se a iteração desse ponto usando a fórmula indicada 2000 vezes. Se depois dessas 2000 iterações o módulo de *z* for maior do que 2, então o ponto é considerado fora do Conjunto de Mandelbrot. Contando-se quantos dos pontos estão dentro do conjunto, pode-se obter uma estimativa da área do conjunto.

Para obter a real experiência de paralelização vamos utilizar o *Cluster* cedido pela universidade utilizando um *nodo* do mesmo e testes com 2, 4, 8 e 16 *threads*, além do sequencial que utiliza apenas uma *thread*. Para a paralelização do conjunto de Mandelbrot é empregada a utilização da *API OPENMP*.

2 ANÁLISE DA PARALELIZAÇÃO

Para a identificação da possibilidade de paralelização o primeiro passo foi entender o código de forma sequencial disponível pelo professor que pode ser visualizado abaixo.

1 Artigo elaborado como primeiro trabalho da disciplina da de Programação Paralela.

2 Aluno de graduação da disciplina de Programação Paralela do curso de Engenharia de Software, da Faculdade de Informática da PUCRS.

3 Aluno de graduação da disciplina de Programação Paralela do curso de Engenharia de Software, da Faculdade de Informática da PUCRS.

4 Professor das disciplinas de Introdução à Ciência da Computação e Programação Distribuída do curso de Ciência da Computação, e da disciplina de Programação Paralela à Engenharia de Software, da Faculdade de Informática da PUCRS.

```
#include <stdio.h>

#include <stdlib.h>
#include <math.h>
#include <omp.h>

# define NPOINTS 2000
# define MAXITER 2000


struct complex{
  double real;
  double imag;
};

int main(){
  int numoutside = 0;
  double area, error, ztemp;
  double start, finish;
  struct complex z, c;

  start = omp_get_wtime();

  for (int i=0; i<NPOINTS; i++) {
    for (int j=0; j<NPOINTS; j++) {
      c.real = -2.0+2.5*(double)(i)/(double)(NPOINTS)+1.0e-7;
      c.imag = 1.125*(double)(j)/(double)(NPOINTS)+1.0e-7;
      z=c;
      for (int iter=0; iter<MAXITER; iter++){
            ztemp=(z.real*z.real)-(z.imag*z.imag)+c.real;
            z.imag=z.real*z.imag*2+c.imag;
            z.real=ztemp;
            if ((z.real*z.real+z.imag*z.imag)>4.0e0) {
              numoutside++;
              break;
            }
      }
    }
  }


  finish = omp_get_wtime();

      area=2.0*2.5*1.125*(double)(NPOINTS*NPOINTS-numoutside)/(double)(NPOINTS*NPOINTS);
      error=area/(double)NPOINTS;

      printf("Area of Mandlebrot set = %12.8f +/- %12.8f\n",area,error);
      printf("Time = %12.8f seconds\n",finish-start);

  }
```

Figura 1 – Código fonte da versão sequencial (programa em C)

2.1 Análise do código

Após realizar a análise do código, concluímos que dos 3 laços existentes no código sequencial, apenas um pode ser paralelizado. O laço mais interno não pode ser paralelizado pois há o comando *break*, já o laço intermediário causaria um número desnecessário de criação de *threads* pois a cada iteração do laço mais externo teria a necessidade de recriar as *threads*, dito isso, o laço escolhido a ser paralelizado é o mais externo pois não apresenta nenhuma restrição para o seu paralelismo.

Cabe ressaltar o tratamento da seção crítica presente dentro do laço mais interno na variável *numoutside*, para com que o valor seja incrementado de forma correta, além das variáveis privadas para cada *thread c*, *z* e *ztemp*.

2.2 API OpenMP

Para a paralelização do código sequencial foi utilizada a *API* (application programming interface) *OpenMP* (Open Multi-Processing). A *API* trabalha com o modelo de execução fork-join e dentre suas vantagens é permitir a paralelização do programa com pequenas modificações no código-fonte.

A utilização do *OpenMP* se dá através de diretivas de compilação iniciadas com *#pragma omp alguns exemplos abaixo:*

- *parallel*: precede um bloco de código que será executado em paralelo

- *for*: precede um laço, for, com iterações independentes que pode ser executado em paralelo

- *parallel* for : uma combinação das primitivas, *parallel*, e, *for*

- *critical* : precede uma seção crítica do código

3 IMPLEMENTAÇÃO

Abaixo a implementação do código paralelo utilizando as diretivas do OpenMP em conjunto com a análise realizada no capítulo anterior.

```
#pragma omp parallel for private(c, z, ztemp)
 for (int i = 0; i < NPOINTS; i++) {
   for (int j = 0; j < NPOINTS; j++) {
     c.real = -2.0 + 2.5 * (double)(i) / (double)(NPOINTS) + 1.0e-7;
     c.imag = 1.125 * (double)(j) / (double)(NPOINTS) + 1.0e-7;
     z = c;
     for (int iter = 0; iter < MAXITER; iter++) {
       ztemp = (z.real * z.real) - (z.imag * z.imag) + c.real;
       z.imag = z.real * z.imag * 2 + c.imag;
       z.real = ztemp;
       if ((z.real * z.real + z.imag * z.imag) > 4.0e0) {
         #pragma omp critical
         numoutside++;
         break;
       }
     }
   }
 }
```

Figura 2 – Trecho de código fonte da versão paralela (programa em C)

Com essa implementação conseguimos alcançar os mesmos resultados do que com a versão sequencial no entanto com um aumento significativo de performance. No próximo capítulo explicaremos de uma forma mais detalhada o ganho de performance.

3.1 Balanceamento de carga

Como o laço mais interno contém o código *break*, existe a possibilidade de algumas *threads* acabarem antes das outras, desta maneira é possível realizar o balanceamento de carga, aplicando em conjunto com a instrução *#pragma omp parallel for private(c, z, ztemp)*, uma nova instrução, *schedule(guided).*

```
#pragma omp parallel for private(c, z, ztemp)schedule(guided)
```

Figura 3 – Trecho de código fonte da versão paralela com balanceamento (programa em C)

4 RESULTADOS

Após a etapa da implementação e obtenção dos resultados utilizando a máquina Grad da PUCRS, a qual já foi mencionada na seção 1(Introdução), com os dados em mãos conseguimos gerar os gráficos para

melhor ilustração da diferença de se paralelizar uma aplicação. O primeiro gráfico mostra o tempo de execução das versões do algoritmo em relação aos NPOINTS a serem realizados. Onde cada uma dessas versões o que muda é a quantidade de processadores deslocados para execução do algoritmo e com isso nosso grafico se comportou da seguinte forma:

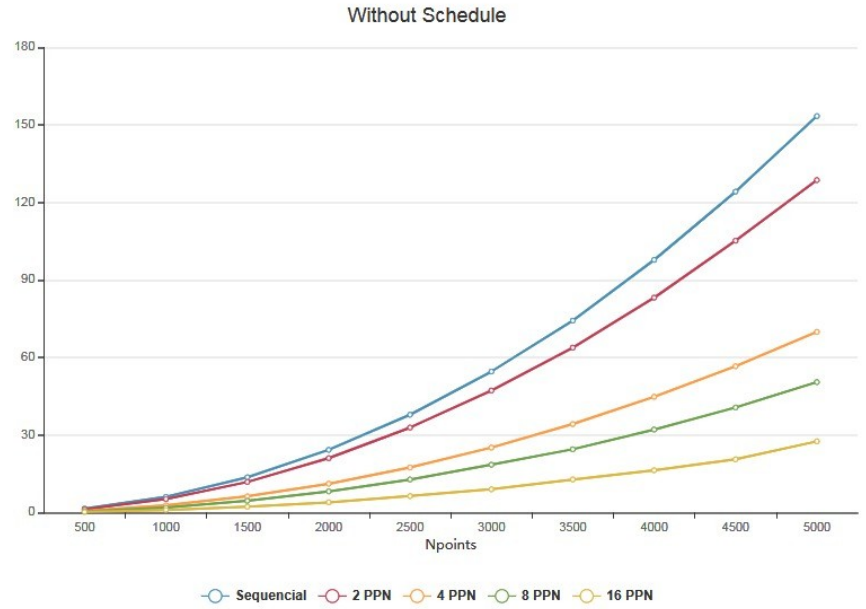

Figura 4 – Resultados sem balanceamento

Outra atividade proposta no trabalho da disciplina era utilizarmos o balanceamento de cargas entre as *threads*, após a mesma execução acima porém com o uso do balanceamento de carga se obteve o seguinte resultado:

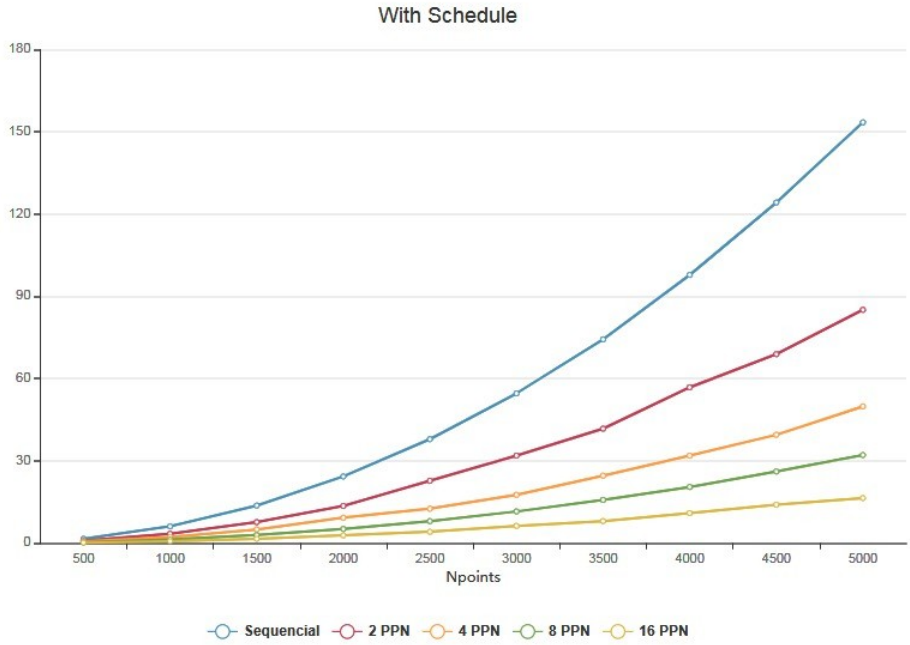

Figura 5– Resultados com balanceamento

Por fim, após as duas análises expostas acima, podemos perceber que chegando ao mesmo resultado do algoritmo sequencial obtivemos uma melhora de aproximadamente 82% utilizando 16 processadores para execução, sem utilizarmos o balanceamento de carga. Já utilizando o balanceamento de cargas obtivemos

uma melhora de aproximadamente 89%.

Dito isso, vamos apresentar um gráfico, o qual, irá apresentar a medida de eficiência do *Speed-Up* em relação ao *Speed-Up* ideal com relação ao numero de *threads*, segue o gráfico:

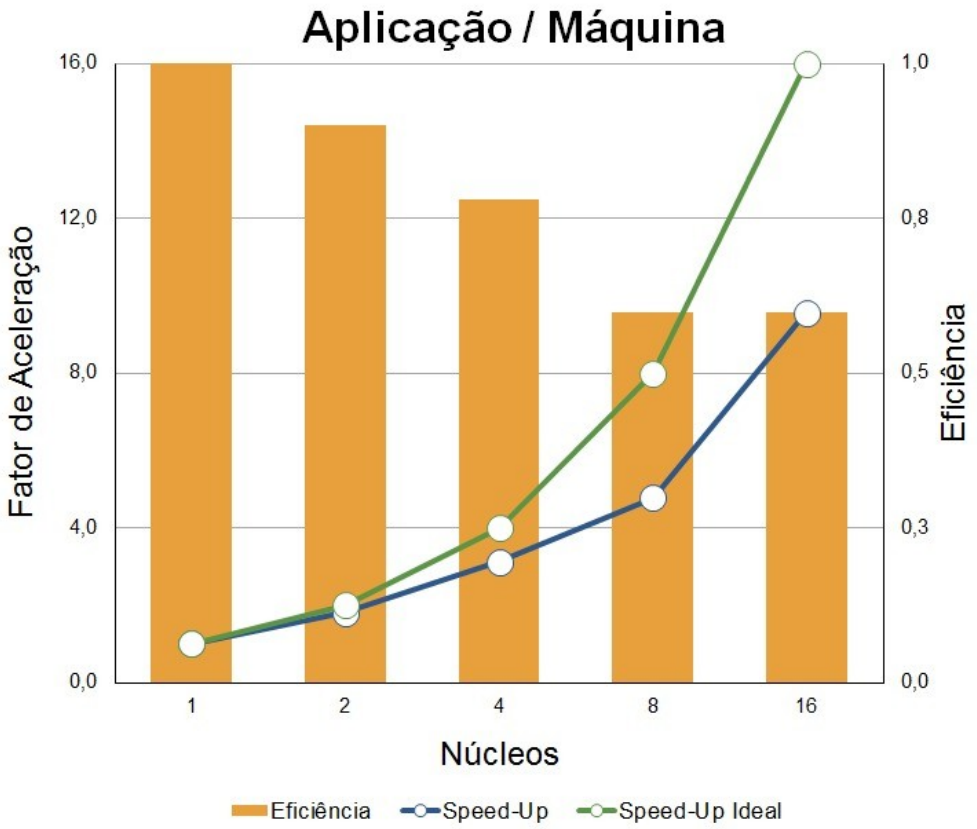

Figura 6 – SpeedUp

Observando a imagem acima, pode-se afirmar que com 2 e 4 *threads* a eficiência do algoritmo é quase ideal chegando perto dos 92%, já na analise com 8 *threads* ela fica inferior a 61% e por ultimo com 16 *threads* fica em 57%, analisando estes valores podemos definir que há uma grande perda na eficiência do algoritmo quando o mesmo é executado com mais de 4 threads.

5 Conclusão

Este artigo apresentou uma sugestão de paralelismo para o código sequencial da Área de um Conjunto de Mandelbrot (BULL, 2019). Além disto também apresentamos os resultados obtidos juntos com a *API OpenMP*, compreendendo as métricas de avaliação do desempenho de um código paralelo. Estimamos que utilizamos em torno de 5 horas dos nodos do LAD para todos os testes.

Podemos concluir que a paralelização impactou de forma positiva e drástica no tempo de execução do algoritmo mesmo com adição de poucas linhas.

Referências

BULL, Mark. Shared Memory Programming with OpenMP - Exercise Notes. [S.l.]: ARCHER, 11 nov. 2019.

Disponível em:

< http://www.archer.ac.uk/training/courses/2019/11/openmp_online/OpenMP-Online-Exercises.pdf >. Acesso em 19 maio 2020.

CONJUNTO DE MANDELBROT. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation,

2020. Disponível em: < https://pt.wikipedia.org/wiki/Conjunto_de_Mandelbrot >. Acesso em: 19 maio 2020