Ánalise de Aplicação de Programação Paralela Através de Estudos de Casos em Implementações em Problemas de Fractais

Ariel F. Azevedo[1], **Rodrigo Acosta. ***[2], **Rodrigo Duarte** [1]

[1]*Laboratory of Ubiquitous and Parallel Systems* (LUPS) -
Universidade Federal de Pelotas (UFPel)

[2]Centro de Desenvolvimento Tecnológico -
Faculdade de Ciência da Computação Universidade Federal de Pelotas (UFPel)

{afazevedo,rmduarte, rdbacosta}@inf.ufpel.edu.br

Resumo. *Este artigo tem como objetivo apresentar uma solução para o problema de fractal (fractus do latim, fração, quebrado) de forma paralela nas linguagens C++11 e OpenMP, bem como uma abordagem sequencial para mostrar a diferença e importância de casos paralelisados.*
Este artigo será composto de uma sessão de introdução para apresentar o problema em si, logo após serão apresentadas ideias e soluções, e por fim resultados obtidos ao longo do trabalho serão debatidos.

1. Introdução

O uso de ambientes de programação paralela e distribuída tem sido recorrente para a execução de aplicações que exijam uma significativa capacidade de processamento de dados .[DONGARRA et al. 2002]. Dentre os principais motivos para utilização de programação paralela, pode-se citar a importância de reduzir o tempo necessários para as aplicações solucionarem determinados problemas e, também, a necessidade de resolver problemas mais complexos de maior dimensão.[ANDREWS 2001]. Inicialmente, motivada pela resolução de problemas com grande relevância científica e econômica, denominados Grand Challenge Problems(GCPs) .[Stout et al. 1998] iniciou-se a utilização de técnicas de processamento paralelo e distribuído.Atualmente, diversas aplicações exigem ou requerem um alto poder de computação,para desta forma,efetuar o processamento de grandes quantidades de informações. O presente artigo busca demonstrar a aplicação de um problema sendo resolvido com o uso de programação paralela. Apesar de ser apenas uma avaliação para um problema que, de certa forma, demanda menor quantidade de processamento que os problemas complexos que temos atualmente, e que fazem uso de programação paralela, como por exemplo, análise de fenômenos climáticos (movimento das placas tectônicas), físicos (órbita dos planetas), químicos (reações nucleares), dentre outros tantos. Este artigo tem como objetivo então tratar sobre a implementação de fractais utilizando as ferramentas C++11 e OpenMP, e está dividido nas seguintes sessões: 1 Introdução, 2 Problema, onde será abordado um pouco sobre que é Fractal e suas características, 3 Conceitos, onde será falado sobre os principais conceitos abordados pelo artigo, para melhor entendimento da parte do leitor, 4 Metodologia, onde será descrito a forma como o problema foi abordado e as soluções tentadas, 5 Resultados onde serão apresentados alguns dos resultados obtidos nos testes realizados e por fim, mas não menos importante 6 Conclusão, onde será então discutido o desfecho do artigo e conclusão do grupo sobre o trabalho e resultados obtidos.

2. Problema

A teoria fractal tem sua origem na descoberta do matemático alemão Karl Weierstrass que encontrou uma função com a propriedade de ser contínua em todo seu domínio, mas em nenhum ponto diferenciável. As plotagens dessas funções eram difíceis, pois elas são recursivas, então o trabalho manual era praticamente impossível. Com o advento do computador o professor Benoît Mandelbrot foi o primeiro a utilizar a máquina para plotar a função recursiva estudada por Pierre Fatou, hoje chamada de Conjunto de Mandelbrot ou simplesmente Fractal de Mandelbrot.

O conjunto de Mandelbrot é definido como o conjunto específico de pontos do plano complexo de Argand-Gauss que obedecem a distância máxima de 2 da origem do plano, isto é, "não tendem ao infinito"para a sequência definida pela recorrência do número complexo $Z = x + yi$. $Z_0 = 0$

$$Z_{n+1} = Z_N^2 + C$$

Onde $Z0$ e Z_{n+1} são iterações n e n+1 do número complexo Z, e $C = a + bi$ fornece a posição de um ponto do plano complexo. Já a parte real e imaginária do complexo Z pode ser desenvolvida até encontrarmos $_{n+1} = x_n^2 - y_n^2 + a$ e $y_{n+1} = 2x_n y_n + b$. Para calcular os pontos de fractal pode-se utilizar o seguinte algoritmo:

Algoritmo 1: ALGORITMO DE MANDELBROT

```
1 int Mandelbrot(Complexo c)
2 int i = 0, ITR = 255;
3 float x = 0, y = 0 tmp = 0;
4 enquanto(x^2 + y^2 <= 2^2 && i < ITR)
5 tmp = x^2 - y^2 + c.real;
6 y = 2 * x * y + c.img;
7 i++;
8 se(i < ITR) retorna i; senão retorna 0;
```

A geração da imagem de um fractal dependerá da quantidade de pontos no domínio, neste caso distância máxima de 2 da origem, e o número máximo de iterações para determinar se o ponto pertence ou não ao conjunto. Para se obter resoluções aceitáveis, isto é, imagens onde é possível observar o padrão de similaridade multi-escala, imagens com resolução maiores que 1200x1200 devem ser utilizadas. Desta forma temos um problema que exige uma grande quantidade de operações em função da resolução do fractal que se deseja obter. A figura 1 mostra um exemplo de fractal.

Tendo isto em vista, utilizamos de programação paralela para tentar reduzir o problema fazendo com que ele seja feito de forma concorrente.

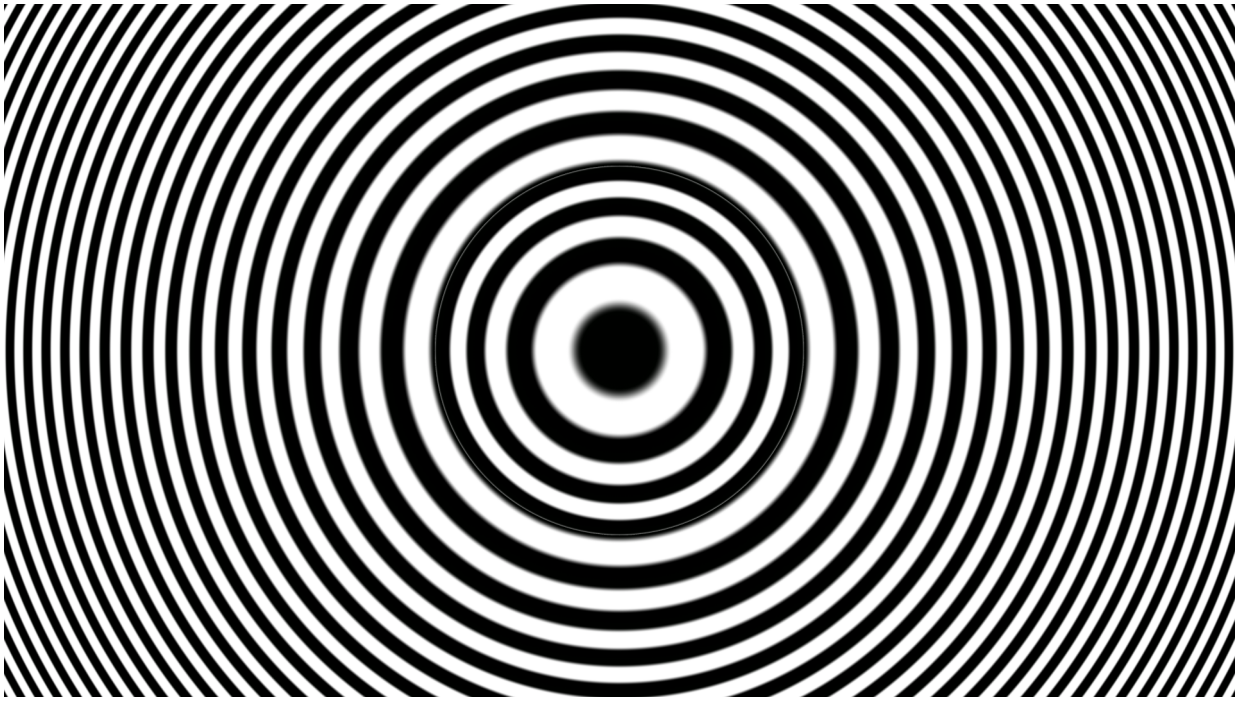

Figura 1. Fractal

3. Conceitos

Nesta sessão será então abordado os principais conceitos utilizados ao longo deste artigo, com um objetivo principal de informar o leitor. Ao decorrer desta sessão espera-se o entendimento básico de tais conceitos para então na próxima sessão começar a especificar o trabalho em si.

Esta sessão está divida em 4 (quatro) subsessões, são estas: Programação Sequencial, Programação Concorrente, C++11 e OpenMP. Seguimos então para a subsessão 1.

3.1. Programação Sequencial

Programação Sequencial (*Top Down*) é aquiela que é executada sequencialmente, da primeira a última linha de código, seguindo rigorosamente a ordem em que o programa foi escrito [Cormen et al. 1990].A figura 2 ilustra este processo, onde um determinado problema é executado sequencialmente, possuindo assim uma série de instruções que devem ser executadas em um único processador.

3.2. Programação Paralela

Um programa é considerado paralelo quando este é visto como um conjunto de partes que podem ser resolvidas concorrentemente [Cormen et al. 1990]. Cada parte é igualmente constituída por uma série de instruções sequenciais, mas que no seu conjunto podem ser executadas simultaneamente em vários processadores, conforme ilustra a Figura 3.

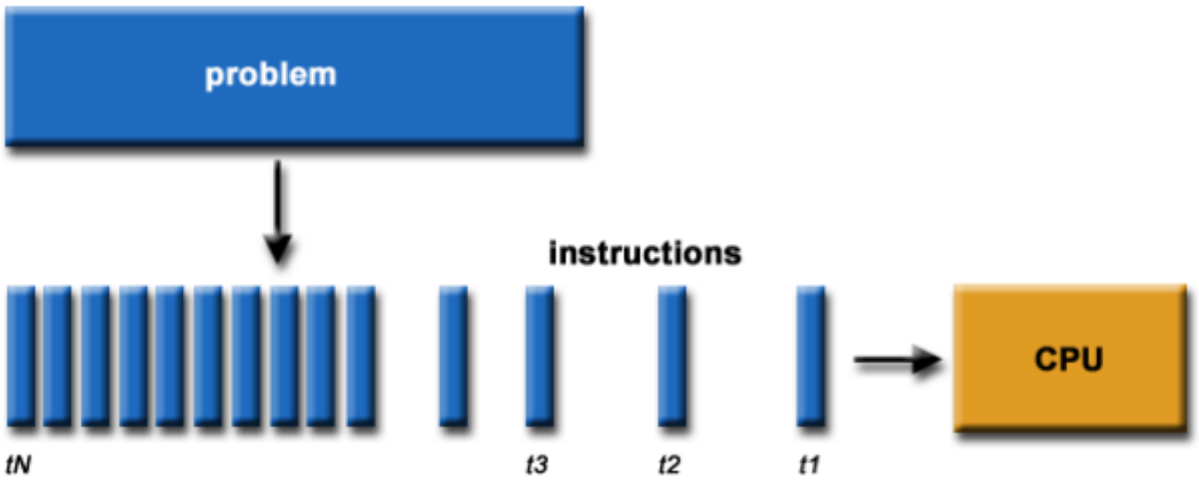

Figura 2. Programação Sequencial

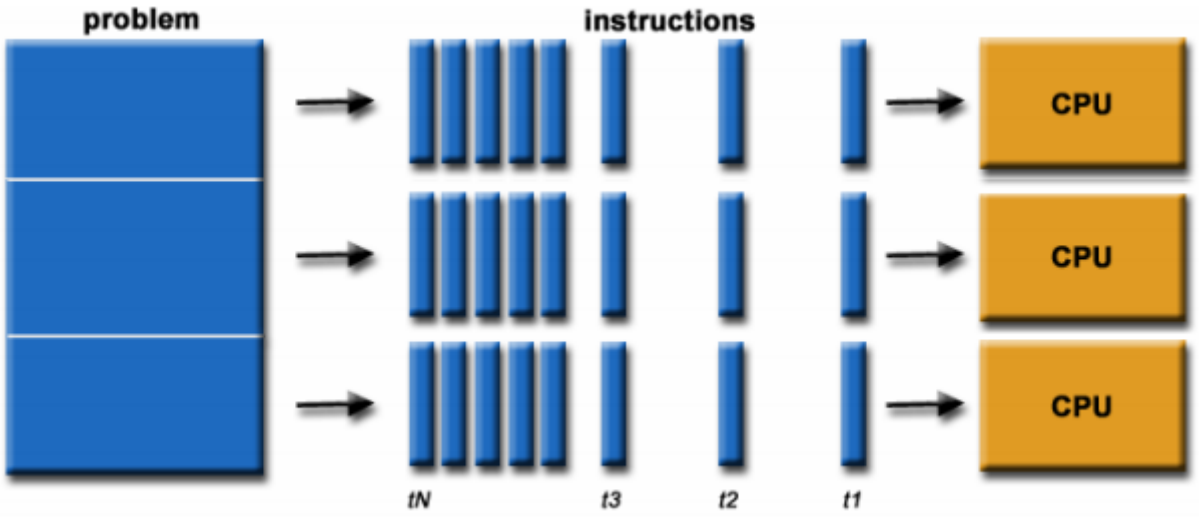

Figura 3. Programação Paralela

3.3. C++11

C++11, formalmente conhecido por C++0x, é uma versão da linguagem de programação padrão C++. Foi aprovado pela ISO em 2011, substituindo C++03 e suplantado pelo C++14 em 2014. O nome segue a tradição de nomear de acordo com o ano de publicação da especificação da linguagem. Essa versão inclue várias mudanças ao núcleo da linguagem e extende a biblioteca padrão do C++. Algumas das principais mudanças são:

- Manter a estabilidade e compatibilidade com C++98 e possivelmente com C.
- Melhorar C++ para facilitar Desenvolvimento de sistemas e bibliotecas, em vez de introduzir novas características úteis apenas para aplicações específicas.
- Fornecer soluções adequadas para os problemas do mundo real.
- Aumenta o desempenho e a capacidade de trabalhar diretamente com hardware.
- Entre outras.

3.4. openmp

OpenMP é uma interface de programação (API), portável, baseada na modelo de programação paralela de memória compartilhada para arquiteturas de múltiplos processadores. [OPENMP 2014] Basicamente é composto por três componentes básicos: diretivas de compilação, biblioteca de execução e variáveis de ambiente. Sua principal facilidade é a existência de um único espaço de endereçamento através de todo o sistema de memória, onde cada processador pode ler e escrever em uma ou todas posições de memória. [OPENMP 2013] O OpenMP está disponível para uso com os compiladores C/C++ e Fortran, podendo ser executado em ambientes Unix e Windows (Sistemas Multithreads) e foi definido (e é mantido), por um grupo composto na maioria por empresas de hardware e software, denominado como OpenMP ARb (Architecture Review Board), que possui diversas empresas, dentre elas: Intel, Copmaq, Sun Microsystems, dentre outras.

4. Metodologia

Conforme especificado na sessão 1, este trabalho foi realizado utilizando o algoritmo de Mandelbrot para problemas de Fractais, o qual foi desenvolvido nas ferramentas C++11, OpenMP e com programação sequencial. Salvo a programação sequencial, ambas as outras ferramentas utilizaram de um número de *threads* entre um e oito, executados dez (10) vezes com cada número de *threads*. Ou seja, para um teste com uma (1) thread, foram realizados dez (10) testes, e assim sucessivamente até os dez testes com oito *threads*. A máquina utilizada para realizar todos os testes, foi a seguinte: CPU:Amd Phenon X6 1055T 2.8Ghz, 6 Pocessadores, 6M cache L3, 512K cache L2, 64K cache L1, mem Ram: 8G DDR3 1066. Foi calculado o *Speed Up* através da seguinte formula:

$$Speedup = \frac{X}{Y}$$

Onde X é o tempo sequencial, e Y é o tempo paralelo, para definir o tempo médio de execução. Assim então seguimos para a sessão 5.

5. Resultados

Após a realização dos testes, foi então gerado os gráficos, como mostra a Figura 4, Figura 5, Figura 6, Figura 7 onde podemos ver a Eficiência medida pela divisão do *SpeedUp* pelo número de *threads* utilizado, e o número de *threads* no eixo X.

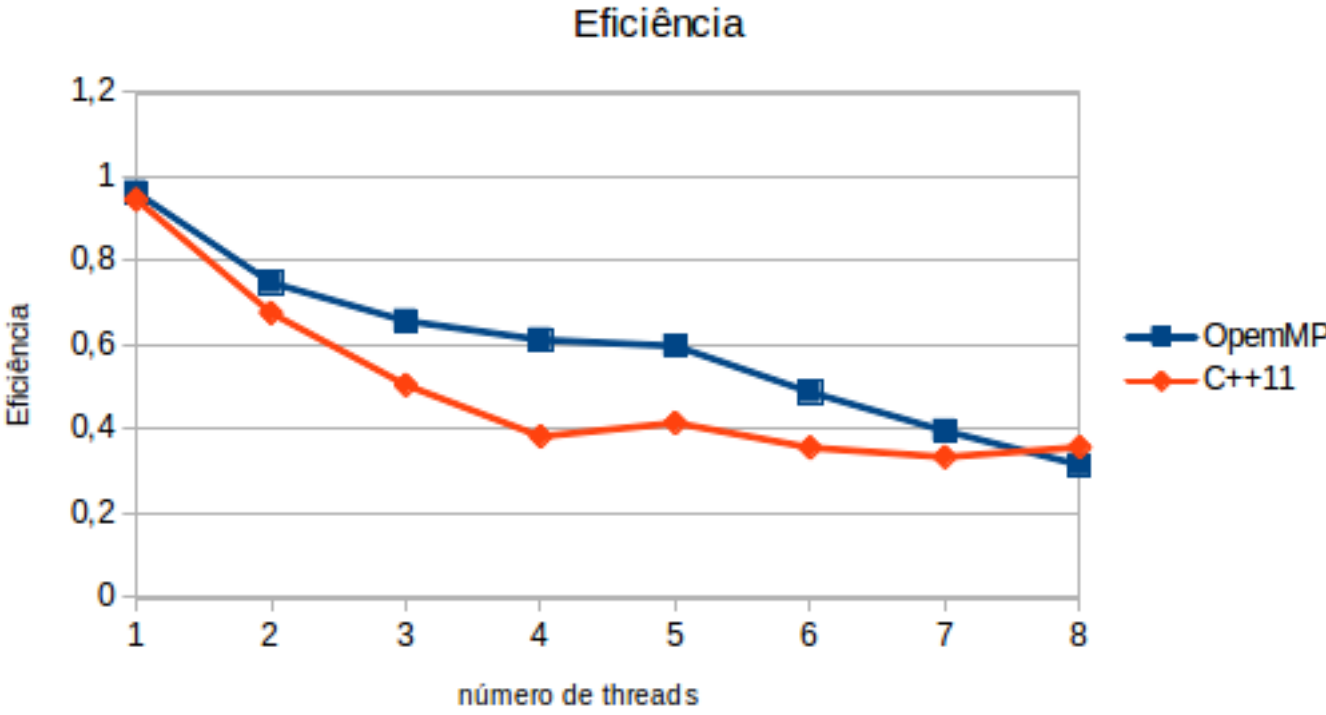

Figura 4. Eficienca 800x600

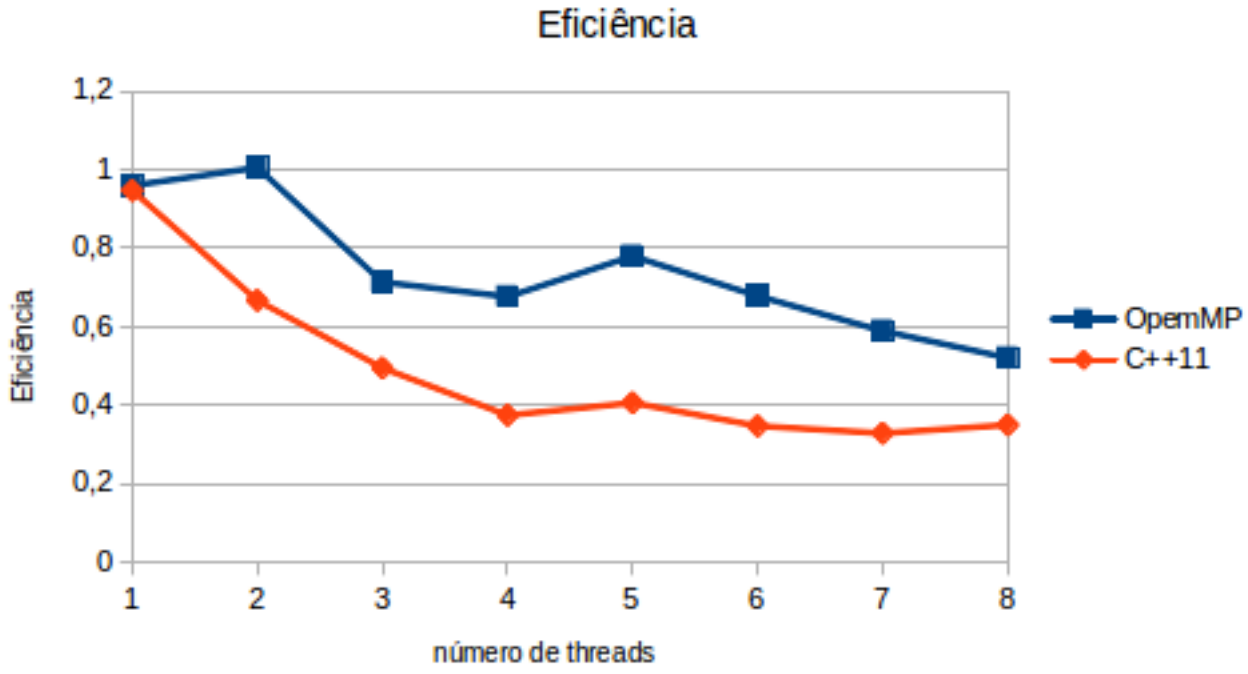

Figura 5. Eficienca para imagens HD

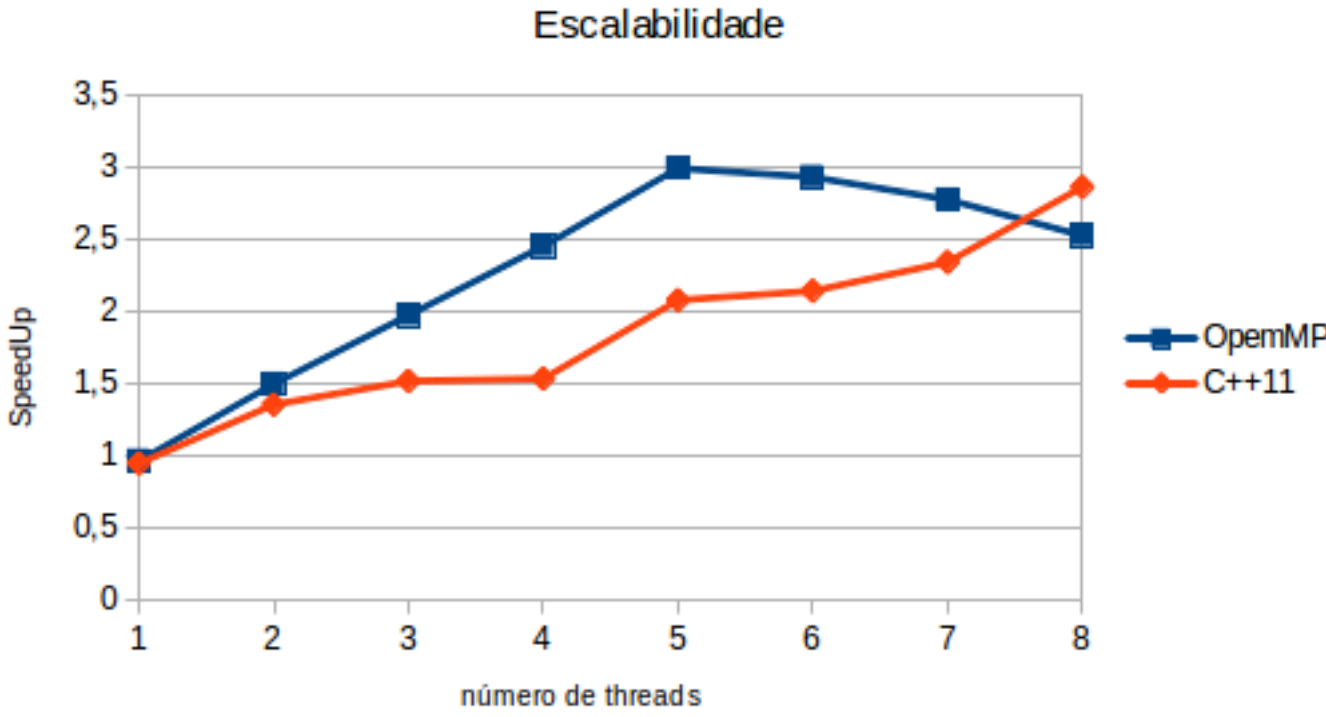

Figura 6. Escalabilidade 800x600

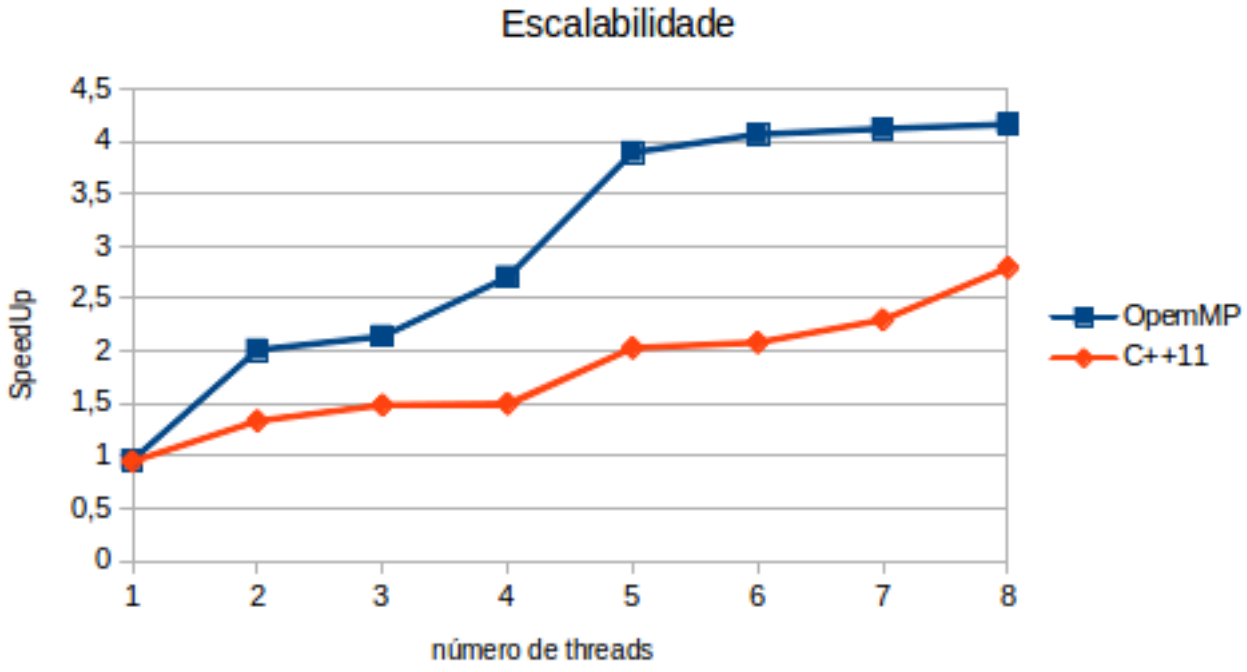

Figura 7. Eficienca para imagens HD

Apesar do grafico de *SpeedUp* nos dizer que estamos ganhando desempenho, e estamos realmente, não é a melhor métrica pois através do Gráfico de Eficiência podemos ver que estamos subaproveitando recursos, ou seja temos mais capacidade de paralelismo do que estamos utilizando.

Isso é normal, pois vários fatores que influenciam no desempenhos como: Cache, Barramento, Memória e até mesmo número de *threads*, pois a partir de certo ponto, deve-se analisar os custos da criação de novas *threads* e em nível mais critico, se o numero de *threads* for muito grande, acabará por serializar a execução.

6. Conclusão

Após então terminarmos os testes, concluímos que OpemMp por ter uma caracteristica de ”Identificar”os setores paralelizaveis, conseguiu ter uma eficiência melhor que C++11 que é necessário ao programador identificar os pontos de paralização e aplicar as técnicas nos locais exatos. Também os graficos estão escondendo uma ilusão de que quyanto mais *threads* melhor, porque não usamos mais de oito *threads*, mas o comportamento das curvas diz que se aumentássemos o número de *threads* C++11 começaria a ganhar desempenho e OpemMp a perder, pode estar relacionando ao custo de estanciar um *thread* em c++11 ser mais leve do que iniciar uma tarefa do OpemMp.

Referências

ANDREWS, G. R. (2001). Foundations of multithreaded, parallel, and distributed programming. page 664p. Addison-Wesley.

Cormen, T., Leiserson, C., and Rivest, R. (1990). McGraw-Hill.

DONGARRA, J., FOSTER, I., FOX, G., GROPP, W., KENNEDY, K., TORCZON, L., and WHITE, A. (2002). *(Ed.). The Sourcebook of Parallel Computing.* [S.l.]: Elsevier.

OPENMP (2013). Simple, portable, scalable smp programming. `https://computing.llnl.gov/tutorials/openMP/`.

OPENMP (2014). The openmp api specification for parallel programming. `http://openmp.org/wp/`.

Stout, Q. F., De Zeeuw, D. L., Gombosi, T. I., Groth, C. P. T., Marshall, H. G., and Powell, K. G. (1998). Adaptive parallel computation of a grand-challenge problem: prediction of the path of a solar-corona mass ejection. In Proceedings of the 1998 ACM/IEEE conference on Supercomputing.