Paradigmas de Processamento Paralelo na Resolução do Fractal de Mandelbrot

Bruno Pereira dos Santos e Dany Sanchez Dominguez

Departamento de Ciências Exatas e Tecnológicas – Universidade Estadual de Santa Cruz (UESC)
16 km Rodovia Ilhéus-Itabuna – CEP 45650-000 – Ilhéus – BA – Brasil

bruno.ps@live.com, dsdominguez@gmail.com

Abstract. *The diverse knowledge area such as engineering, bioinformatics or science needs a great computational power to solve their problems. In this subject, parallel processing is the best solution, this paper has the objective to compare the paradigms of parallel processing (Distributed and multicores CPUs and GPU) to resolve a known mathematical problem, the plot of Mandelbrot's Fractal, when a fractal image need to be produced in a high resolution (1200x1200 pixel) or higher, traditional techniques of sequential processing are overly slow. So option will be presented to solve this problem.*

Resumo. *As várias áreas do conhecimento como engenharias, bioinformática ou ciências cada vez mais necessitam de grande poder computacional para resolução de seus problemas. Neste âmbito, o processamento paralelo é a principal solução. Este artigo tem como objetivo o comparativo dos paradigmas de processamento paralelo (CPUs distribuídos, multicores e GPU) na resolução de um conhecido problema matemático, a plotagem do fractal de Mandelbrot, quando se quer produzir uma imagem de fractal em alta resolução (1200x1200 pixels) ou superior, técnicas tradicionais de processamento sequencial são excessivamente lentas. Como alternativa, serão apresentadas três implementações paralelas para resolver este problema em arquiteturas paralelas diferentes.*

1. Introdução

A solução computacional utilizada tradicionalmente para resolver problemas é a técnica sequencial baseada na máquina de Von Neumann, onde um programa, para este dispositivo, é uma sequência de instruções executadas em um processador. Já um programa que utiliza o paradigma paralelo é visto como um conjunto de partes que podem ser resolvidas concorrentemente sendo estas partes executadas serialmente e concomitantemente em vários processadores. Com base nestes conhecimentos, diversas áreas do conhecimento estão utilizando o paralelismo para resolver seus problemas, como na simulação de exploração de reservatórios de hidrocarbonetos [M. Santos, Dominguez e Orellana 2009], problemas matemáticos [P. Santos, Dominguez e Orellana 2011], assim como bioinformática e engenharias dentre outras [Aiping D, 2011], [Alonso P, 2009], [GoddekeD, 2007].

No que diz respeito ao processamento paralelo destacam-se três principais enfoques, dentre eles, dois aproveitam a unidade central de processamento (CPU) e um as unidades de processamento gráfico (GPU) como fontes de poder computacional. Das técnicas que utilizam a CPU, uma é caracterizada como modelo de memória distribuída

onde vários conjuntos de {CPUs, Memória} distintos cooperam na realização da tarefa, neste trabalho, lançou-se mão da tecnologia MPI (*Message Passing Interface*) [Open-MPI 2012]que fornece ao usuário uma interface amigável possibilitando a comunicação entre os processos distintos. A outra é singularizada pelo fato de sua memória ser compartilhada, onde se utiliza OpenMP (Open MultiProcessing) [OpenMP 2012] que é uma API(Application Programming Interface) que é utilizada para programas em arquitetura multi-processada, isto é, vários *cores* numa máquina compartilhando a mesma memória. Já para o processamento em GPU será abordada a recente tecnologia CUDA (Computing Unified Device Architecture) [CUDA Zone 2012] criada pela nVidia como extensão de linguagens de programação de alto nível como C, C++ e Fortran, para a criação de códigos que executem na GPU como proposto em [GPGPU 2012].

O problema computacional, aqui abortado, é um problema matemático conhecido como Fractal, estes são célebres, pois além de serem classificados como funções recursivas, são de difícil plotagem. Em específico será tratado neste trabalho o Fractal de Mandelbroth [Mandelbrot, Jones, Evertsz e Gutzwiller 2012], o primeiro Fractal plotado por um computador. Quando se quer gerar imagens de um fractal em resolução suficiente para que seja visível o seu padrão de similaridade multi-escala, imagens com resoluções superiores a 1200x1200 pixel devem ser utilizadas. Desta forma gera-se um problema que exige uma grande quantidade de operações em função da resolução desejada para o fractal.

O objetivo, neste artigo, é dar ênfase ao comparativo entre os diferentes paradigmas paralelos como executado em trabalhos anteriores [1.P. Santos, Dominguez e Orellana 2011] e [2.P. Santos, Dominguez e Orellana 2011] para implementação do Fractal de Mandelbrot e o tempo de processamento para diversas resoluções (1024x1024, 2048x2048, 4096x4096, 8192x8192 e 16384x16384), assim como o SpeedUP de cada implementação. O que permitirá a realização de inferências e a expansão das conclusões dos trabalhos pretéritos sobre os principais pontos fortes e fracos de cada técnica de paralelização. Além de agregar conhecimento técnico e de recursos humanos com capacidades de desenvolvimento nas diversas formas de arquiteturas paralelas ao NBCGIB (Núcleo de Biologia Computacional e Gestão de Informações Biotecnológicas da UESC) [NBCGIB 2012].

Para realizar os experimentos foram feitas diversas implementações do algoritmo para gerar o fractal de Mandelbroot: uma versão serial utilizando a linguagem C, uma versão paralela empregando C e técnicas de programação multi-processada OpenMP, uma variante paralela tirando proveito de C e recursos de programação de troca de mensagens MPI, e uma versão aplicando C e as soluções disponibilizadas pela extensão CUDA para processamento em GPU. A fim de compará-las quanto ao seu tempo e SpeedUP na geração das imagens de fractal de alta resolução.

Na próxima sessão deste artigo será apresentado o problema computacional adotado neste comparativo, para logo após exibir as principais características de cada modelo paralelo. Em seguida serão expostos e discutidos os resultados obtidos. Finalmente apresentam-se as conclusões da pesquisa e possíveis melhorias para trabalhos futuros.

2. Fractal de Mandelbrot

A descoberta de Karl Weierstrass (1815 - 1897), matemático alemão, introduzia a teoria do fractal. Basicamente um fractal é uma função recursiva que contém uma propriedade interessante: funções que são contínuas em todo seu domínio, no entanto em nenhum ponto é diferençável. A plotagem desse tipo de função manualmente, na época, era impraticável visto que sua característica recursiva dificultava o trabalho.

Pierre Fatou (1878 - 1929), também matemático, apresentou um conjunto de pontos do plano complexo definidos por: $z = z^2 + c$. Benoît Mandelbrot (1924 - 2010) usou esse conjunto e um computador para a primeira plotagem do fractal, hoje chamado de Conjunto de Mandelbrot ou simplesmente Fractal de Mandelbrot.

Um conjunto específico de pontos do plano complexo de Argand-Gauss é definido como o Fractal de Mandelbrot, estes pontos devem obedecer à distância máxima 2 da origem do plano, ou seja, "não tendem ao infinito" para o encadeamento definido pela recorrência do número complexo $Z = x + yi$, sendo x a parte real e y a parte imaginária, temos:

$$(1) Z_0 = 0$$

$$(2) Z_{n+1} = Z_n^2 + C$$

Onde Z_0 e Z_{n+1} são iterações n e $n + 1$ do número complexo Z, e $C = a + bi$ é a posição de um ponto do plano complexo que se deseja iterar. Desenvolvendo a parte real e imaginária de Z obtemos (3) e (4):

$$(3) \quad x_{n+1} = x_n^2 - y_n^2 + a$$

$$(4) y_{n+1} = 2x_n y_n + b.$$

Para desenvolver os códigos paralelos e o serial tirou-se proveito das equações (3) e (4) para a construção do algoritmo que representa o Conjunto de Mandelbrot. Este algoritmo é apresentado na Figura 1:

```
int conj_mandelbrot(complexo c){
    int I = 0; ITR = 255;
    float x = 0; y =0; tmp = 0;
    enquanto(x² + y² < 2² && i < ITR){
        tmp = x² - y² + c.real;
        y = 2 * y * x + c.img;
        i++;
    }
    Se (i<ITR) retorne i;
    Senão retorne 0;
}
```

Figura 1 – Algoritmo para geração do conjunto de Mandelbrot.

A imagem do fractal a ser produzida dependerá dadistância máxima da origem |2|representada por ($x^2 + y^2 < 2^2$); a quantidade de pontos em seu domínio, isto é, o tamanho da matriz que simulará o plano complexo; e o número máximo de iterações (ITR = 255)que da um "refinamento" na precisão de pertinência ou não do ponto ao conjunto de Mandelbrot, ou seja, quanto maior é o número ITR maior o refinamento.

3. Principais características OpenMP, MPI e CUDA

Nesta sessão aborda-se o método de construção de cada versão paralela implementada bem como os principais recursos utilizados das respectivas APIs. Não será apresentada a versão serial, visto que sua construção é trivial tendo em mãos o algoritmo da Figura 1.Na implementação serial aloca-se uma matriz para representar o plano complexo e para cada elemento desta matriz utiliza-se a função descrita pelo algoritmo.

3.1 OpenMP

OpenMP fornece ao usuário uma interface de programação para criar aplicações multiprocessadas com memória compartilhada, estes programas podem utilizar as linguagens C, C++ e Fortran. Basicamente as construções em OpenMP são feita por meio de diretivas de pré-processamento que informam ao compilador que aquela região é paralela e assim possibilitando a criação dos threads que executarão o trecho paralelo [Figura 2].

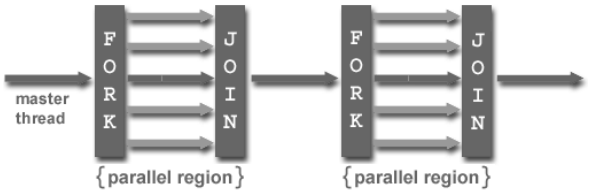


Figura 2 – Fluxo de execução em implementações com OpenMP.

Na implementação, utilizou-se uma diretiva no loop principal onde se itera cada ponto da matriz que representa o plano complexo. Em específico usou-se o #pragma_omp_parallel_for, que é empregado especificamente para loops do tipo for para executar modelo de programação conhecido como SPMD (Single ProgramMultiple Data)[SMPD Programming 2012]. Em conjunto com a diretiva, foram aplicadas as opçõesschedule(dynamic, 3) que define como as iterações serão dividas entre os threads, dynamics informa que as iterações serão divididas em blocos de 3 para cada thread, sendo assim após o termino de seus afazeres o thread "pede" mais trabalho. Também foi tirado proveito das opções shared e firstprivate para indicar variáveis compartilhadas e qual a forma de inicialização respectivamente.

Vale ressaltar a facilidade de implementar códigos paralelos usando OpenMP, visto que a construção pode ser feita em cima da versão serial adicionando-se diretivas (#pragma_omp), com suas variantes e opções obtém-se um código paralelo.

3.2 MPI

Na modalidade de computação paralela, em que se estabelece comunicação entre vários processadores ou dentro de um cluster de computadores, pode-se utilizar o padrão MPI, que disponibiliza os mecanismos necessários para desenvolver o entendimento entre os nodos atuantes. Sendo assim a aplicação construída poderá se dividir em um ou mais

processos, que trocam mensagens, cada processo recebe uma copia do programa a ser executado e por meio de seu rank (Identificador do processo) executa uma parte do código, por isso MPI é conhecido como MPMD (Multiple Program Multiple Data).

Por ser uma modalidade de computação paralela baseada em troca de mensagens entre os diversos processos e seu comando é dado por instruções de controle como: if(rank = 0) faça A, if(rank = 1) faça B. A coordenação do fluxo de execução é relativamente complexa e geralmente envolve barreiras e pontos de sincronização. Por outro lado quando se fala em hardware para esse tipo de enfoque paralelo, podem-se utilizar computadores domésticos de baixo custo tornando a escalabilidade da maquina paralela mais barata.

A construção do código paralelo utilizando MPI para plotagem do fractal de Mandelbroot consistiu em utilizar um processo (rank = 0), como responsável por distribuir os pontos do plano complexo entre todos os processos envolvidos, e por receber os resultados de cada processo para gerar a imagem resultante. Sendo assim cada nó é responsável pela construção de uma parte da imagem. Para alcançar um bom balanceamento da carga entre os processos, a distribuição do plano complexo foi feita em listras verticais.

A implementação de códigos paralelos usando MPI é uma tarefa complexa, onde a distribuição de tarefas e a coordenação do fluxo de execução estão sob a responsabilidade do programador.

3.3 CUDA

Nesta sessão apresenta-se as principais características da tecnologia CUDA, que motivou os trabalhos anteriores [1.P. Santos, Dominguez e Orellana 2011] e [2.P. Santos, Dominguez e Orellana 2011]. CUDA é uma plataforma da nVidia que implementa a GPGPU (General Purpose Computationon Graphics Hardware) [GPGPU 2012] ela aumenta de forma significante o desempenho computacional através da arquitetura paralela contida em suas GPUs. Como extensão da linguagem C, C++ e Fortran esta plataforma diminui a curva de aprendizado na programação de GPUs [CUDA Zone].

Por ser uma extensão, CUDA oferece uma API para identificar threads disparados pela GPU para execução do código; bem como qualificadores que podem indicar onde será executado o código; em qual memória residirá aquela variável; e funções para transporte entre memória principal (RAM) e memória da GPU.

A aplicação desenvolvida com CUDA aproveitou-se da maneira da organização dos threads na GPU [Figura 3], onde um kernel (unidade de processamento paralelo) possui um grid, que contém blocos e estes possuem threads. Então como a organização se expande de forma bidimensional, podemos representar uma imagem facilmente utilizando os threads no plano.

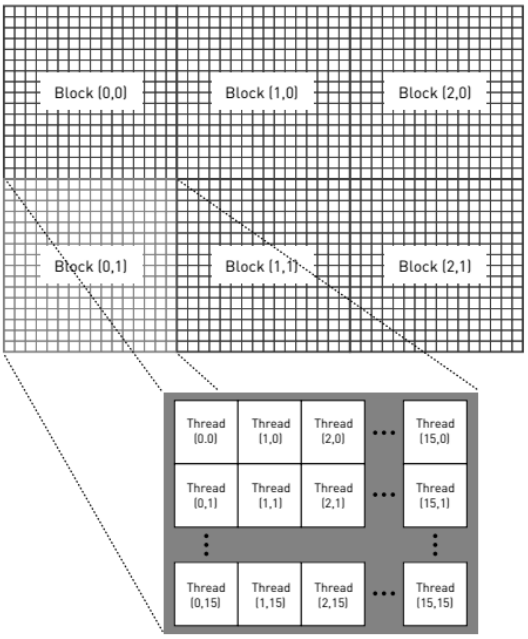


Figura 3 – Fluxo de execução em implementações com CUDA [Sanders e Kandrot].

Desta forma, a criação de aplicações que utilizem a plataforma CUDA depende da aprendizagem da extensão, bem como da estrutura de threads disponíveis sua organização [Figura 3]. Adicionalmente deve existir hardware habilitado para CUDA [CUDA GPUs 2012] da nVidia.

4. Resultados Obtidos

Para realização dos experimentos numéricos nas versões dos códigos serial, OpenMP e CUDA foi utilizada uma estação de trabalho descrita na Tabela 1. Esta estação está equipada com placa nVidia 9800GT, vale ressaltar que das placas aceleradoras gráficas habilitadas para CUDA esta é uma das que apresenta menos recursos. Para realizar os experimentos da versão MPI um cluster foi empregado com a configuração exibida na Tabela 2, estas estações se encontram no NBCGIB.

Tabela 1 – Configuração da estação de trabalho para experimentos com as versões serial, OpenMP e CUDA.

Configuração da estação de trabalho 1	
Processador	Intel ® Core i7 CPU 860 2,8GHz
Memória RAM	8GB
Placa Gráfica	GPU Nvidia GeForce 9800GT, 112 cores, 512 de RAM, 256bits PCI Express 16x

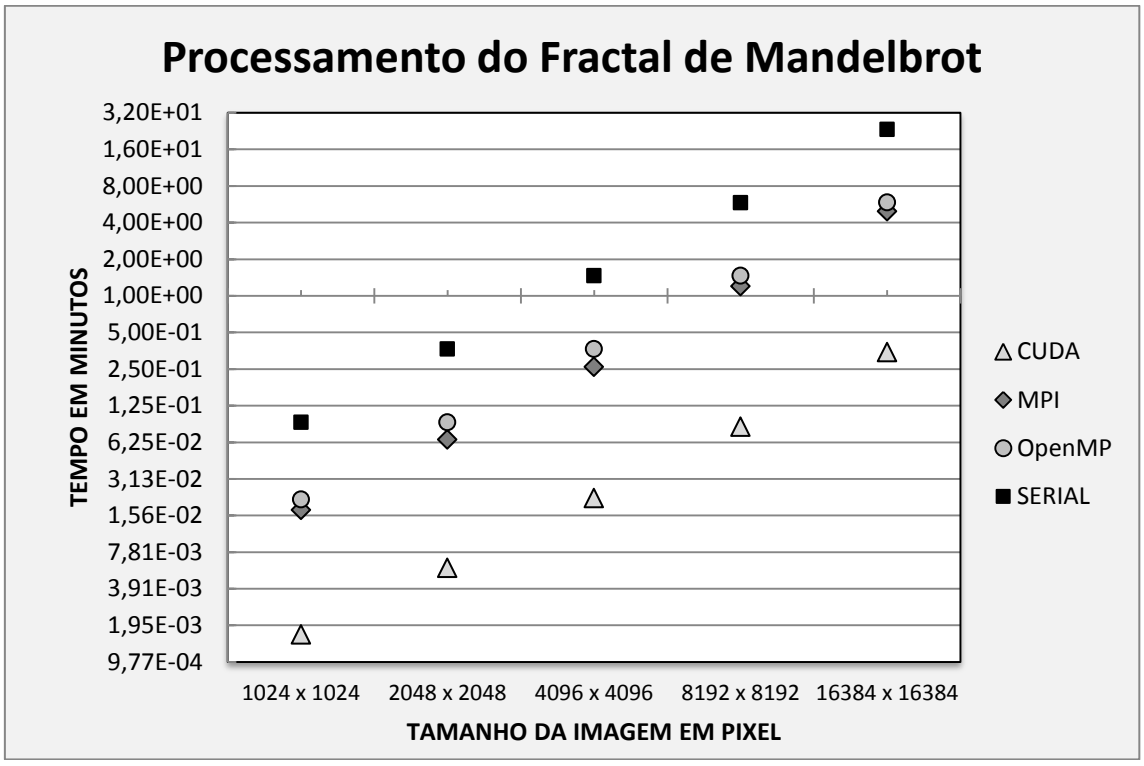
Tabela 2 – Configuração da Estação de Trabalho para experimentos com a versão MPI.

Configuração da estação de trabalho 2	
Processadores	8 nós Genuine Intel ia-64, modelo Madison com 9M cachê
Memória RAM	16GB Compartilhada

Para gerar os executáveis de todas as versões utilizou-se os compiladores nvcc(Cuda compilation tools, release 4.0, V0.2.1221) disponibilizado pela nVidia em [CUDA Downloads 2012] para geração de código para suas placas gráficas, mpicc(mpicc for MPICH2 version 1.4.1p1) para compilar o fonte em MPI e gcc(version 4.3.6) para as versões serial e OpenMP. Destaca-se que em todas as versões foram compiladas com a otimização nível dois (-O2), com isso obtém-se um código mais enxuto e eficiente.

Finalmente, após a compilação os executáveis foram submetidos a uma bateria de tomadas de tempo, na geração de imagens do fractal em alta resolução, estas variaram em diversos tamanhos partindo de 1024x1024 pixels até 16384x16384 pixels. Para os vários experimentos realizados está exposto no Gráfico 1 o tempo de processamento de cada implementação em função do tamanho da imagem. Em seguida no Gráfico 2 é apresentado o SpeedUP, ou seja a razão entre o tempo de processamento do algoritmo serial e o tempo de processamento do algoritmo paralelo. Esta razão fornece um indicador do quão rápido foi a implementação paralela em relação ao serial. Vale ressaltar que foram utilizados 4096 como grau de refinamento, isto é, o numero de máximo de iterações.

Gráfico 1 – Tempo de processamento das versões e diversos tamanhos de imagem



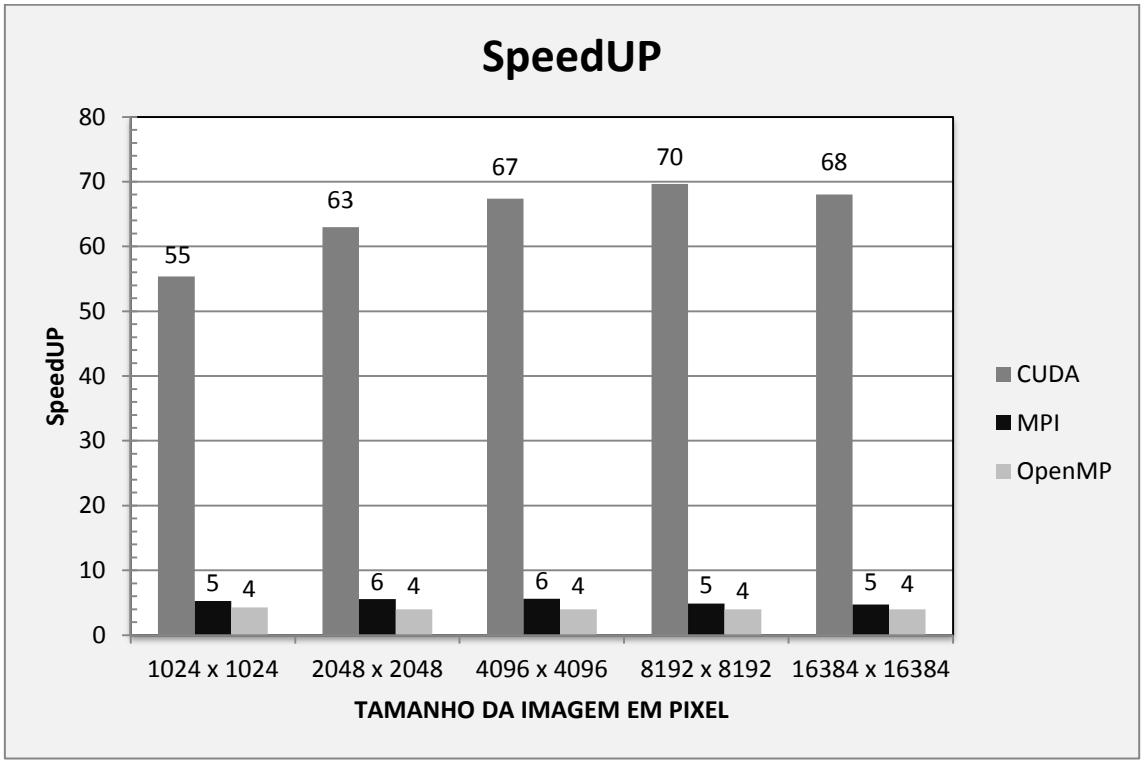


Gráfico 2 – SpeedUP de processamento das versões em diversos tamanhos de imagem.

Admitindo que para obter os tempos do Gráfico 1, afere-se somente tempo de processamento do segmento paralelo dos códigos, assim realmente é possível comparar o tempo de processamento concorrente das versões.

Como constatado no Gráfico 1 o executável serial foi o mais lento, especialmente na maior resolução de 16384 x 16384 pixels, quando comparado com as versões paralelas. Entre as versões paralelas OpenMP e MPI apresentam um comportamento semelhante, com tempo crescente em função do tamanho da imagem, sendo a versão OpenMP ligeiramente superior. A versão CUDA apresentou tempos muito pequenos para todos os tamanhos de imagens analisados.

No Gráfico 2 é observado um SpeedUP elevado para a versão executada naplaca aceleradora gráfica (GPU), sendo esta implementação muito superior as outras implementações. A versão OpenMP obteve um SpeedUP praticamente constante, e versão MPI obteve o melhor desempenho para fractais de média resolução.

5. Conclusão e Trabalhos Futuros

Notou-se que as versões paralelas obtiveram bom desempenho na construção do fractal de Mandelbroot com alta resolução. Ficando a implementação em CUDA com o melhor desempenho, isto se deve ao fato da grande quantidade de *cores* existentes na GPU (vide. Tabela 1), a sua arquitetura paralela que melhora substancialmente algoritmos altamente paralelizáveis, como é o caso do problema computacional abordado, e a baixa transferência de dados necessária entre a CPU e a GPU. No que diz respeito às aplicações MPI e OpenMP, a primeira obteve um SpeedUp um pouco superior que a segunda, isto pode ser explicado pela configuração da estação de trabalho utilizada em que os processadores do cluster são mais robustos, mesmo tendo a latência da comunicação entre os processos.

Não é falso inferir que o processamento paralelo em GPU utilizando CUDA, apesar de ser uma técnica recente, é uma grande fonte de poder computacional, especialmente quando a aplicação é altamente paralelizável, possibilitando o desenvolvimento de programas paralelos com baixa curva de aprendizado, menor custo e espaço físico necessário para o hardware. Um ponto baixo é a necessidade de uma placa aceleradora ou outro dispositivo com GPUs de um fabricante específico, neste caso a nVidia.Porém existem iniciativas, como o frameworks OpenCL (Open Computing Language) [OpenCL 2012], visam a padronização de escrita de programas para GPUs de qualquer fabricante.

No que tange as implementações paralelas tradicionais obtiveram melhor desempenho do que a versão serial, devido ao alto grau de independência dos dados, caracterizando um alto paralelismo. Sendo estas, boas alternativas a serem exploradas nas arquiteturas paralelas convencionais, sejam de memória compartilha ou distribuída.

Para trabalhos futuros podem-se melhorar as implementações paralelas, utilizando melhores práticas, como as apresentadas em [NVIDIA Corporation 2012] bem como construir versões híbridas compostas por OpenMP e MPI, OpenMP e CUDA, MPI e CUDA, e MPI CUDA e OpenMP, para novos comparativos com mais abrangências as diversas implementações possíveis. Também se encontra em desenvolvimento uma versão paralela em CUDA referente à problemática abordada em [M. Santos, Dominguez e Orellana 2009], a fim de obter um comparativo dos paradigmas paralelos solucionando o dado problema.

7. Referencias

Aiping Ding, Tianyu Liu, Chao Liang, Wei Ji, and X George Xu (2011) "EVALUATION OF SPEEDUP OF MONTE CARLO CALCULATIONS OF SIMPLE REACTOR PHYSICS PROBLEMS CODED FOR THE GPU/CUDA ENVIRONMENT".

Alonso, P., Cortina, R, Martínez-Zaldívar, F. J.,Ranilla, J. (2009) "Nevilleelimination on multi- and many-core systems: OpenMP, MPI and CUDA, J. Supercomputing", in press, doi:10.1007/s11227-009-0360-z, SpringerLink Online Date: Nov. 18.

CUDA Downloads, NVIDIA Corporation (2012). Disponível em:

http://developer.nvidia.com/cuda-downloads, março.

CUDA Zone, NVIDIA Corporation (2012). Disponível em: http://www.nvidia.com/object/cuda_home_new.html, março

CUDA GPUs, NVIDIA Corporation (2012). Disponível em:

http://developer.nvidia.com/cuda-gpus, março.

GPGPU (2012). Disponível em: http://gpgpu.org/, março.

Goddeke D, Strzodk R, Mohd-Yusof J, McCormick P, H.M Buijssen S, Grajewski M. e

Turek S. (2007) "Exploring weak scalability for FEM calculations on a GPU-enhanced cluster".

M. Santos, Adriano. e Dominguez, Dany S. e Orellana, Esbel T. V. (2009) "Utilizando MPI na Paralelização do Método de Decomposição de Domínio para Resolução Numérica da Equação de Poisson"

Mandelbrot, B. Jones, P.W.Evertsz, C.J.G. e Gutzwiller, M.C. “Fractals and chaos : the Mandelbrot set and beyond “

NVIDIA Corporation, (2011) “NVIDIA OpenCL Best Practices Guide”. Disponívelem:http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/OpenCL_Best_Practices_Guide.pdf, março.

NBCGIB, Universidade Estadual de Santa Cruz – UESC (2012). Disponível em:

http://labbi.uesc.br/nbcgib/fr/home, março.

Open-MPI (2012). Disponível em:http://www.open-mpi.org/software/ompi/v1.4/, março.

OpenMP (2012). Disponível em: http://openmp.org/wp/, março.

OpenCL (2012). Disponível em: http://www.khronos.org/opencl/, março.

1.P. Santos, Bruno. e Dominguez, Dany S. e Orellana, Esbel T. V. (2011) “O Problema Do fractal de Mandelbrot como Comparativo de Arquiteturas de Memória compartilhada – GPU vsOpenMP”

2.P. Santos, Bruno. e Dominguez, Dany S. e Orellana, Esbel T. V. (2011) “Aplicando Processamento Paralelo com GPU ao Problema do Fractal de Mandelbrot”

SMPD programming (2012).Disponível em:

http://www.openmp.org/presentations/sc98/mod4_pregions/sld014.htm, março.

Sanders, J. e Kandrot, E. “CUDA by Example: An Introduction to General-Purpose GPU Programming”