Uma Proposta de Aumento de Desempenho na Simulação de Árvores Trinomiais para Precificação de Opções

Marcelo Lisboa Rocha
Curso de Ciência da Computação – UFT
109 Norte, Av. NS 15, ALCNO 14. Bloco II, Sala 21. Palmas-TO-Brasil. Cep: 77001-090
marcelolisboarocha@yahoo.com.br

Giordano Bruno Rodrigues Machado
Curso de Ciência da Computação – Centro Universitário UNIRG
Avenida Antônio Neves da Silva S/N, Pq. das Acácias, Gurupi - TO, CEP: 77400-000
giordano@gmail.com

Alessandro Rodrigues e Silva
Curso de Ciência da Computação – Centro Universitário UNIRG
Avenida Antônio Neves da Silva S/N, Pq. das Acácias, Gurupi - TO, CEP: 77400-000
mestre.alessandro.s@gmail.com

RESUMO

Este trabalho tem por finalidade demonstrar os benefícios da utilização de Unidades de Processamento Gráfico (GPU) para implementar um software na área financeira para cálculos de árvores trinomiais para precificação de opções e comparar o desempenho desta execução em relação ao desempenho mesmo método rodando em uma Unidade Central de Processamento (CPU). Este trabalho também apresenta uma análise de eficiência quanto ao ganho de desempenho com a utilização de GPU's em substituição às CPU's.

PALAVARAS CHAVE. Árvores Trinomiais, Precificação de Opções, GPU.

ABSTRACT

This work has as goal show the benefits of using Graphic Processor Unit (GPU) to implement a software in financial area to calculate trinomial trees for options pricing and compare the performance of this execution related to the performance of the same method running on a Central Processing Unit (CPU). This work also presents an analysis of efficiency about performance gain with the use of GPUs against CPUs.

KEYWORDS. Trinomial Trees, Options Pricing, GPU.

Main Area: High Performance Computing.

1. Introdução

Existe uma crescente necessidade de desenvolver soluções para o tratamento, a recuperação e a disseminação de informação relevante a partir de volumes exponencialmente crescentes de dados. Para tratar esses grandes volumes de dados e informação distribuída é também essencial a exploração eficiente de todos os níveis de paralelismo, do nível do chip, hoje com processadores multicore, passando pelo nível de arquitetura dos nós integrados através de redes de alto desempenho em um cluster, até a formação de grades (grids) pela comunicação de clusters heterogêneos através de uma rede clássica, tipo internet (SBC, 2006).

A computação de alto desempenho é extremamente útil na área de finanças auxiliando na resolução de problemas que são definidos em modelos financeiros sob a forma de seqüências de cenários e a probabilidade de realização dos mesmos. A evolução dos preços das ações e as taxas de juros são freqüentemente definidas dessa forma. A partir de estado inicial conhecido e abrangendo um horizonte temporal de múltiplos futuros períodos, esses modelos assumem a forma de um cenário de árvores. Como um exemplo, avaliação dinâmica da probabilidade de três ativos financeiros no período de 10 anos, cada um destes sendo descrito em graus e as probabilidades de subir ou descer seus respectivos preços dentro de um ano, resulta em cenário em forma de árvore com mais de um bilhão (23^{10}) de nós terminais. O tempo de computação necessária para a resolução desses problemas pode se estender pelo período de horas ou até dias, daí a utilização de computação paralela para acelerar o retorno de uma solução em tempo útil faz-se necessária (MORITSCH, 2006).

Várias ferramentas foram desenvolvidas com o intuito de auxiliar nos processos de paralelização dos cálculos exigidos por sistema dessa natureza. Aqui será apresentada a biblioteca CUDA™ (*Compute Unified Device Architecture*) que se propõe a disponibilizar o poder de processamento das GPU's (*Graphics Processing Units*) para processamento genérico na solução de muitos problemas computacionais complexos em frações de tempo da execução em uma CPU (*Central Processing Unit*).

A arquitetura CUDA™ da NVidia® suporta a linguagem C, uma das linguagens de alto nível mais amplamente utilizadas, para desenvolvimento de aplicações de alta performance. Esta arquitetura apresenta excelente resultado quanto ao desempenho na execução de problemas mais complexos de computação (ZONE, 2009). Serão expostos, a seguir, os conceitos relacionados ao contexto onde o trabalho foi inserido.

O contrato futuro é um acordo estabelecido para comprar ou vender um ativo em determinada data no futuro a preço previamente estabelecido. Existem diversas bolsas negociando esses contratos no mundo (HULL, 2005).

Métodos que realizam simulações para aplicação aos mercados financeiros são alternativas extremamente intensivas do ponto de vista computacional, pois exige que seja feito um conjunto de simulações para cada dia da amostra de preços do ativo objeto (SOUZA, 1999). Dentre esses algoritmos, existe um método numérico de avaliação conhecido com "Árvores Trinomiais". A árvore trinomial é uma técnica que tem como base a projeção dos diferentes valores que o preço de um ativo pode atingir durante sua vida. O volume de cálculos utilizados no processamento das árvores trinomiais é grande e para torna eficiente sua aplicação, os métodos mais eficientes de computação disponíveis tornam-se necessários.

Neste trabalho foram utilizadas unidades de processamento gráfico (GPUs) para a execução de aplicações genéricas, onde foi desenvolvida uma aplicação para resolução de árvores trinomiais,

uma técnica aplicada na área de finanças. Desta forma, foi implementado um protótipo para demonstração das vantagens do uso de GPUs em relação ao uso de CPUs tanto em tempo de processamento quanto em custo. Também foi realizado um estudo comparativo entre a forma de programação em CUDA™ e OpenMP, que também faz uso de paralelismo.

De modo a demonstrar a viabilidade da utilização de GPU para resolver problemas que demandam alto poder computacional, foi implementado um software para o cálculo de preços de opções utilizando árvores trinomiais e a API CUDA™. O desenvolvimento da aplicação se deu com a intenção de se alcançar maior desempenho em relação às alternativas disponíveis no mercado.

O restante deste trabalho está organizado como se segue. A próxima seção apresenta questões básicas de opções como também a sua precificação via árvores trinomiais. Na seqüência, apresenta conceitos básicos a respeito de GPUs, inclusive da que foi utilizada neste trabalho, como também sobre a API CUDA que permite explorar o poder computacional das GPUs. Na seção 3 é descrita a implementação computacional do modelo de árvore trinomial, tanto seqüencial quando paralela (utilizando a GPU). Já na seção 4 são apresentados os testes e análises dos resultados computacionais obtidos das versões seqüências e paralelas do método proposto. As conclusões e trabalhos futuros são apresentados na seção 5.

2. Referencial Teórico

Aqui serão apresentados conceitos básicos de opções e como as mesmas podem ser precificadas utilizando o método da árvore trinomial. Posteriormente serão mostradas informações a respeito da arquitetura e funcionamento de GPUs e como a API CUDA disponibiliza este recurso para o desenvolvedor obter ganho de desempenho na aplicação.

2.1. Opções no Mercado Financeiro

Os mercados futuros remontam à Idade Média. Foram originalmente desenvolvidos para satisfazer as necessidades de produtores e comerciantes de produtos agrícolas.

Profissionais da área de finanças têm dedicado enorme esforço para compreender o processo de formação dos preços de títulos negociados em mercados de capitais. Para isso, foram elaborados modelos de avaliação que apresentam bom grau de confiabilidade (LEMGRUBER, 1995).

Contratos sobre opções são fundamentalmente diferentes de outros tipos de contratos futuros negociados em bolsas de valores. Uma opção dá a seu detentor o direito de fazer algo e não a obrigação que existente nos outros tipos. Em uma compra de uma opção, um pagamento inicial é obrigatório (HULL, 2005).

Existem dois tipos básicos de opções, as opções de compra (*calls*) e as opções de venda (*puts*). Uma opção de compra dá ao seu (titular da opção) o direito de comprar um ativo por certo preço em determinada data. Na opção de venda, o titular obtém o direito de vender o ativo por determinado preço em data futura. O preço combinado entre as partes em um contrato é conhecido como preço de exercício (*stike price*) e a data acordada para a realização deste é chamada de data de vencimento, de exercício ou de maturidade (HULL, 2005).

As opções negociadas nos mercados europeus podem ser exercidas somente na data de maturidade definida no contrato. As opções do tipo americanas permitem o exercício de forma antecipada limitando-se à data de vencimento. Uma opção garante ao titular um direito, mas não a obrigação de exercê-la (HULL, 2005).

É de extrema importância citar que existem seis fatores que influenciam os preços das opções, sendo eles (HULL, 2005): preço a vista da ação (S_0), preço de exercício (X), prazo até a data de vencimento (T), volatilidade do preço da ação (σ), taxa de juro livre de risco (r), e dividendos e bonificações esperados durante a vida da opção (em alguns casos). Estes fatores servem de base para os mais variados modelos para precificação de opções, onde o mais famosos é o *Black and Scholes* (LEMGRUBER, 1995). Maiores detalhes a respeito do modelo de *Black and Scholes* pode ser vistos no livro de HULL (HULL, 2005).

2.2. Árvores Trinomiais

A árvore trinomial é uma técnica muito útil para apreçar as opções sobre ações. Essa técnica trata-se de um diagrama que representa os diferentes caminhos que podem ser seguidos pelo preço de uma ação durante a vida de uma opção (GROSSO, 2006).

O modelo trinomial é um modelo onde o preço das opções sobre as ações é monitorado por sucessivos pequenos períodos de tempo, desde a data onde é realizada o contrato até o seu vencimento. A cada passo, o preço pode assumir três novos valores realizando movimento de subida *up*, manutenção do valor ou movimento de descida *down* com as respectivas probabilidades de ocorrência P_u, P_m e P_d. Todas as possibilidades que os preços das ações podem atingir eventualmente formam uma estrutura de uma árvore trinomial.

Cada nó da árvore trinomial tem três nós filhos cujos valores são $u.S$, S e $d.S$, sendo S_0 o valor na raiz é que corresponde ao valor da ação no tempo 0. Já u e s representam respectivamente a volatilidade do valor na subida (*up*) e na descida (*down*) no tempo. A **Ilustração 1** apresenta a estrutura de uma árvore trinomial.

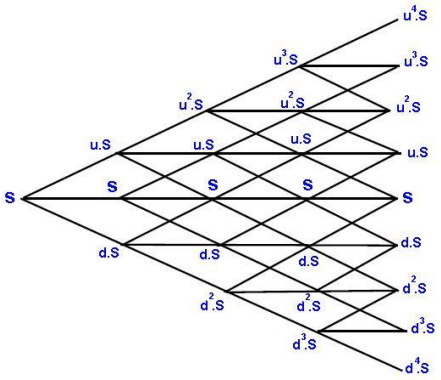

Ilustração 1 - Árvore trinomial de quatro passos.

O modelo de precificação trinomial é baseado no modelo *Black & Scholes* que é um modelo de variação de preços contínuo no tempo (LEMGRUBER, 1995). O modelo de *Black Scholes* permite calcular *Pu*, *Pm*, *Pd*, *u* e *d*, conforme especificadas nas equações de 1 a 5.

$$u = e^{\sigma\sqrt{3\Delta T}} \qquad d = e^{-\sigma\sqrt{3\Delta T}}$$

Equação 1. **Equação 2.**

onde ΔT representa a diferença de tempo entre dois passos consecutivos na árvore. Note que $u.d = 1$, e após N passos as folhas da árvore trinomial possuirá (2N + 1) nós. Desta forma, é possível calcular as probabilidades utilizando o modelo de *Black and Scholes*:

$$Pu = \frac{\Delta T(\sigma^2 + \mu(\mu.\Delta T + \sigma\sqrt{3.\Delta T}))}{6.\sigma^2.\Delta T} \qquad \text{onde } \mu = r - \frac{\sigma^2}{2}$$

Equação 3.

$$Pd = \frac{\Delta T(\sigma^2 + \mu(\mu.\Delta T - \sigma\sqrt{3.\Delta T}))}{6.\sigma^2.\Delta T} \qquad \text{onde } \mu = r - \frac{\sigma^2}{2}$$

Equação 4.

$$Pm = 1 - Pu - Pd$$

Equação 5.

Tendo realizada a representação trinomial, será visto a seguir como é possível calcular o valor justo para as opções no tempo 0. É necessário calcular o valor em cada nó da árvore e o prêmio pela opção será o valor encontrado na raiz da árvore.

Para o cálculo do valor no nó V_n utiliza-se a fórmula especificada na equação 6. Os preços encontrados nos nós $V_{d,\ n+1}$, $V_{m,\ n+1}$ e $V_{u,\ n+1}$ são os preços dos três nós filhos do novo valor. Pelo modelo de *Black and Scholes* todos os valores da folhas podem ser gerados através da fórmula apresentada na equação 7.

$$V_n = (P_d.V_{d,n+1} + P_m.V_{m,n+1} + P_u.V_{u,n+1}).e^{-r.\Delta T}$$

Equação 6.

$$S_n = S_0.e^{i.\sigma\sqrt{3.\Delta T}}$$, onde i varia de $-n$ até n

Equação 7.

Na computação existem muitas formas de lidar com os algoritmos para resolução dos problemas da formação de preços de títulos. Entretanto, esses algoritmos requerem um grande poder computacional. O uso de computação paralela se faz necessária onde o volume de cálculos é grande e fazem-se necessárias respostas com rapidez.

2.3. Arquitetura da GPU

Uma GPU (Unidade de Processamento Gráfico) é composta por vários multiprocessadores (variando atualmente numa faixa de 4 a 30). Um multiprocessador é composto por 8 núcleos de processadores escalares com a função de criar, gerenciar e executar operações de threads concorrentes. Para gerenciar as centenas de threads na execução de vários programas, o multiprocessador emprega uma nova arquitetura conhecida por SIMT (*Single Instruction Multiple Thread*) baseada na arquitetura SIMD (Single Instruction Multiple Data). O multiprocessador mapeia cada thread para um núcleo do processador escalar, e cada thread executa independentemente suas instruções. O multiprocessador SIMT cria, gerencia, agenda e executa threads em grupos de 32 threads paralelas chamadas *warps*.

2.4 CUDA

Com CUDA™, a GPU pode ser definida como um processador SIMD com grande poder de paralelismo, sendo limitado apenas pela quantidade de memória disponível no hardware gráfico (HARISH, 2007). A instalação do kit para desenvolvimento fornece várias ferramentas para o uso da linguagem C. Uma questão importante a considerar a respeito de CUDA™ é que a mesma foi projetada para trabalhar sobre uma arquitetura escalável para os multiprocessadores disponíveis na

GPU, possibilitando assim o aumento no ganho de desempenho da aplicação desenvolvida.

3. Resolução de Árvores Trinomiais na GPU

Nesta seção será analisada a técnica que foi utilizada para o desenvolvimento do programa para o cálculo de precificação de opções em GPU, culminando na realização da paralelização da resolução dos cálculos intrínsecos às árvores trinomiais. Para atingir esse objetivo foi necessário desenvolver um esquema mais robusto de controle de threads para uma eficiente manipulação dos dados nas diferentes áreas de memória existentes em uma GPU.

3.1 Resolução na CPU

Os cálculos em árvores trinomiais utilizando CPUs podem ser facilmente realizados através de algoritmos que operam de forma serializada. Inicialmente são gerados os valores das folhas da árvore utilizando o método de *Black and Scholes* visto na **Equação 7**, conforme mostrado na **Ilustração 2**.

Em seguida, realizando sucessivas reduções no vetor, executa-se o movimento de subida da árvore voltando nos períodos de tempo, utilizando as expressões expostas na **Equação 3**, **Equação 4**, **Equação 5** e **Equação 6**, até que o valor na raiz da mesma seja encontrado. O algoritmo é demonstrado na **Ilustração 3**.

```
variável:
    i, Número de passos: inteiro
    S, X, vDt, vetor_opção[ ( Número de passos * 2 ) + 1 ] : real

    S ← Preço do ativo à vista
    X ← Preço de exercício
    vDt ← Efeito da volatilidade

Procedimento: valorNoVencimento ( real: A, real: B, real: C, inteiro: D )
início
    real E ← A * e (D * C) – B
    Se E > 0 então
        retorne E
    Senão
        retorne 0
    Fim se
fim Procedimento


Para i ← 0 até 2 * Número de passos, faça
    vetor_opção[ i ] ← valorNoVencimento ( S, X, vDt, ( i – Número de passos ) )
fim Para
```

Ilustração 2 - Cálculo dos valores no vencimento para uma call na CPU.

Para demonstrar as vantagens quanto ao ganho de desempenho na resolução dos cálculos intrínsecos às árvores trinomiais foram necessárias comparações dos tempos de execução do algoritmo nas CPU e GPU. As medições foram realizadas através da inserção de funções para monitoramento do tempo de execução dos algoritmos no código do programa e os resultados apresentados em gráficos e tabelas.

```
variável:
    i, j, Número de passos: inteiro
    vetor_opção, probSubidaR, probEstavelR, probDescidaR : real

probSubidaR ← probabilidade de subida
probEstavelR ← probabilidade de manter nível
probDescidaR ← probabilidade de descida


Para i ← 0 até Número de passos, faça
        Para j ← 0 até ( 2 * Número de passos ) - 2 faça
            vetor_opção[ i ] ← probSubidaR * vetor_opcão[ j + 2 ] + probEstavelR * vetor_opcão [ j + 1 ] + probDescidaR * vetor_opcão [ j ]
        fim Para
fim Para
```

Ilustração 3 - Redução da árvore trinomial na CPU.

3.2 Paralelização do Algoritmo na GPU

Com o intuito de realizar um maior aproveitamento dos recursos computacionais existentes na GPU, foi realizada uma divisão das tarefas a serem executadas em paralelo.

Num primeiro momento são gerados todos os dados necessários para a computação da árvore na CPU e realizada a transferência destes do *host* para o *device*. Na GPU é feita a paralelização dos cálculos de todos os possíveis valores das opções no vencimento, onde cada multiprocessador se encarrega deste trabalho em uma opção.

Os vetores onde são armazenados os valores dos preços no vencimento são declarados de forma que seus tamanhos sejam sempre múltiplos de 16. Esse artifício é utilizado para manter a coerência dos dados na memória provendo ganho de desempenho nas operações de escrita e leitura. O esquema é demonstrado na **Ilustração 4**:

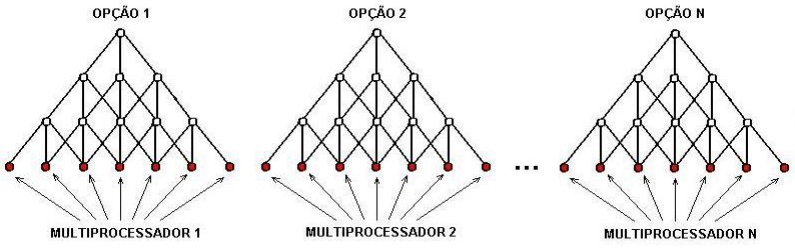

Ilustração 4 - Cálculo dos valores no vencimento na GPU.

Nas **Ilustração 5** e **Ilustração 6** tem-se os algoritmos que descrevem os algoritmos utilizados na execução dos cálculos dos valores no vencimento das opções e da resolução das árvores trinomiais respectivamente na GPU.

A fim de paralelizar a execução do algoritmo sobre toda a estrutura da árvore de preços foi necessária a realização de uma decomposição geométrica da mesma. Essa decomposição se fez necessária devido ao fato de o algoritmo desenvolvido realizar a transferência dos dados para área de memória *shared* que, possui limitação no armazenamento. A memória *shared* da GPU possui capacidade máxima de armazenamento de aproximadamente 4000 valores de ponto flutuante, tamanho esse que é facilmente excedido em execuções de árvores trinomiais de muitos passos. Como exemplo, uma árvore cuja avaliação abranja um número de 2000 passos terá um vetor contendo os dados no vencimento com 4001 valores de ponto flutuante. Essa transferência de dados ocorre com o intuito de aumentar os ganhos de desempenho na execução do programa.

```
ProcedimentoDEVICE: valorNoVencimento ( real: A, real: B, real: C, inteiro: D )
início
   real E ← A * e^(D*C) – B
   Se E > 0 então
      retorne E
   Senão
      retorne 0
   Fim se
fim Procedimento

Kernel: CalcularVencimentoGPU ( )
   Referenciar endereço de memória para valores no vencimento no bloco atual: inicio
      Opção ← endereço opção do bloco atual [ ( 2 * Número de passos ) + 16 ]
   fim referenciar
   variável:
      IDTHREAD ← THREADIDX.X
      A ← Valor atual do ativo para a opção atual
      B ← Preço de exercício para a opção atual
      C ← Volatilidade por variação de tempo para a opção atual

   Para i ← IDTHREAD até ( 2 * Número de passos ) passos Tamanho do Cache, faça
      Opção[ i ] ← valorNoVencimento ( A, B, C, ( i – Número de passos ) )
   fim Para
fim Kernel

ALGORITMO NO HOST:
Cópia dos dados de todas as opções do HOST para DEVICE: início
   Opções DEVICE ← Opções HOST
fim cópia

CHAMADA AO KERNEL:
CalcularVencimentoGPU <<< Número de opções, Tamanho do Cache >>> ( )
```

Ilustração 5 - Algoritmo para cálculo do vencimento na GPU.

```
Kernel: ExecutarArvoreTrinomialGPU ( )
   Referenciar endereço de memória para valores no vencimento no bloco atual: inicio
      Opção ← endereço opção do bloco atual [ ( 2 * Número de passos ) + 16 ]
   fim referenciar

   variável:
      OpçãoX [ Tamanho do Cache ]: real (MEMÓRIA SHARED DA GPU)
      OpçãoY [ Tamanho do Cache ]: real (MEMÓRIA SHARED DA GPU)
      IDTHREAD ← THREADIDX.X
      A ← Valor atual do ativo para a opção atual
      B ← Preço de exercício para a opção atual
      C ← Volatilidade por variação de tempo para a opção atual
      PROBS ← Probabilidade de Subida para a opção atual
      PROBM ← Probabilidade de Manutenção para a opção atual
      PROBD ← Probabilidade de Descida para a opção atual
      i, j, m, CACHE_A, CACHE_B, CACHE_C: inteiro

      Para i ← (Número de passos * 2) até maior que 0 passos Variação CACHE, faça
         Para CACHE_A ← 0 até maior que i passos Salto do CACHE, faça
            CACHE_B ← mínimo (Tamanho do CACHE - 1 , i - CACHE_A)
            CACHE_C ← CACHE_B - Variação CACHE

            Sincronizar THREADS
            Se (IDTHREAD <= CACHE_B)
               OpçãoX [ IDTHREAD ] ← Opção [ CACHE_A + IDTHREAD ]

            Para m ← (CACHE_B - 1) até maior que CACHE_C, faça
               Sincronizar THREADS
               Se (IDTHREAD < m)
                  OpçãoY [ IDTHREAD ] ← PROBS*OpçãoX [ IDTHREAD+2 ] +
                                        PROBM*OpçãoX [ IDTHREAD+1 ] +
                                        PROBD*OpçãoX [ IDTHREAD]
               m ← m - 1

               Sincronizar THREADS
               Se (IDTHREAD < m-1)
                  OpçãoX [ IDTHREAD ] ← PROBS*OpçãoY[ IDTHREAD+2 ] +
                                        PROBM*OpçãoY [ IDTHREAD+1 ] +
                                        PROBD*OpçãoY [ IDTHREAD]
               m ← m - 3
            fim Para
            Sincronizar THREADS
            Se (IDTHREAD < CACHE_C)
               Opção [ CACHE_A + IDTHREAD ] ← OpçãoX [ IDTHREAD ]
         fim Para
      fim Para
      VALOR DA OPÇÃO ← OpçãoX [ 0 ] ( PARA O BLOCO ATUAL )
fim Kernel
ExecutarArvoreTrinomialGPU <<< Número de opções, Tamanho do Cache >>> ( )
Cópia dos dados das opções do DEVICE para HOST: início
   Opções HOST ← Opções DEVICE
fim cópia
```

Ilustração 6 - Algoritmo para resolução de árvores trinomiais na GPU.

A estrutura dos dados da árvore trinomial possui um padrão originalmente recursivo e a decomposição geométrica copia “pedaços” da mesma para a memória *shared*. Os dados copiados estão armazenados de forma contínua na memória *global* da GPU. Na execução do algoritmo foram testados vários tamanhos para os “pedaços” com a intenção de se verificar qual a melhor configuração que permitisse um aumento de desempenho, não sendo observadas diferenças significativas. O acesso à memória *shared* provê ganho de tempo com fator de 100x a 150x superior ao tempo de acesso da memória global (FARBER, 2008).

Carregados os valores das folhas é realizado um loop computando os dados da árvore até o instante da data zero. Como os dados são divididos em “pedaços”, ocorre que o loop provoca uma redução de dois nós da árvore em cada passo que é realizado, ou seja, para N passos realizados em direção à raiz, observamos a redução de 2 x N nós da árvore. Depois de realizado o armazenamento

dos dados em *cache*, são computados os valores de um determinado número de passos e em seguida o "pedaço" seguinte da árvore é carregado

Para decomposição da estrutura da árvore a definição de alguns parâmetros foi utilizada. Os pedaços da árvore armazenados na memória *shared* para ganho de desempenho foram denominados tamanho do cache. Esse parâmetro também definiu o número de threads por bloco. Ao final do laço onde é realizada a subida dos níveis da árvore ocorre a redução da quantidade de nós para o nível em questão. O tamanho dessa redução de nós foi chamada de delta_cache. A diferença entre o tamanho do cache e o delta_cache é exatamente o tamanho do pulo realizado pelo algoritmo para o carregamento de outro pedaço da árvore para a memória *shared*. Foram também necessárias as definições dos parâmetros que marcaram o a base do pedaço da árvore em execução, a posição de início do delta_cache e a posição final do delta_cache. Tal procedimento é visualizado na **Ilustração 7**.

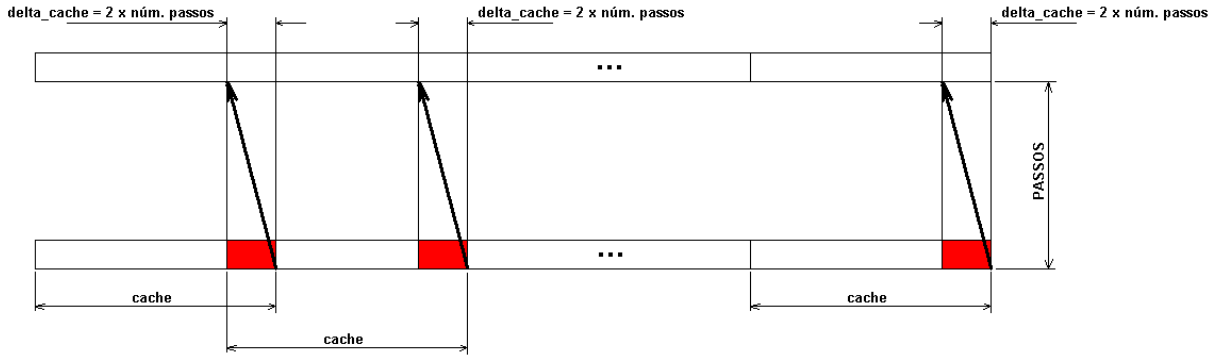

Ilustração 7 - Redução da árvore na GPU.

4. Testes e Análises

O hardware utilizado para os testes computacionais foi composto por um processador Intel® Core™ 2 Duo E8400 3 GHz com 3Gb de memória RAM, placa aceleradora gráfica modelo GeForce 8800 GTS 512 de fabricação da NVidia®. A placa de vídeo foi utilizada devido ao fato de a API CUDA™ dar suporte apenas às placas da geração 8 ou superiores. A placa mãe modelo P5K Premium da Asus foi utilizada nos testes. Essa placa foi escolhida por possuir 2 slots PCI Express com velocidade 16x. Esses slots foram necessários devido à necessidade da instalação de uma segunda placa de vídeo para utilização com o monitor liberando os recursos da GeForce 8800 GTS para a execução dos testes.

A plataforma utilizada para o desenvolvimento do programa de precificação de opções foi composta pelo compilador Visual C++ 2005 Express Edition SP1 escolhido por ser de distribuição gratuita e oferecer suporte a multi linguagens. A instalação foi realizada sobre o sistema operacional Windows XP 32bit Professional. Foram instalados também o CUDA™ Toolkit e o CUDA™ SDK ambos versão 2.0 para Windows 32 bit.

Nos testes foram utilizadas 512 árvores com 1024, 2048, 4096, 8192 e 16384 passos cada, para comparação de desempenho em diversas situações. Os dados dos 512 ativos necessários para avaliação dos valores justos das opções na data 0 foram gerados de forma aleatória.

No **Gráfico 1** são apresentados os tempos (em segundos de CPU/GPU) que foram medidos durante a execução dos cálculos em árvores trinomiais na CPU e a comparação do desempenho do algoritmo paralelizado na GPU.

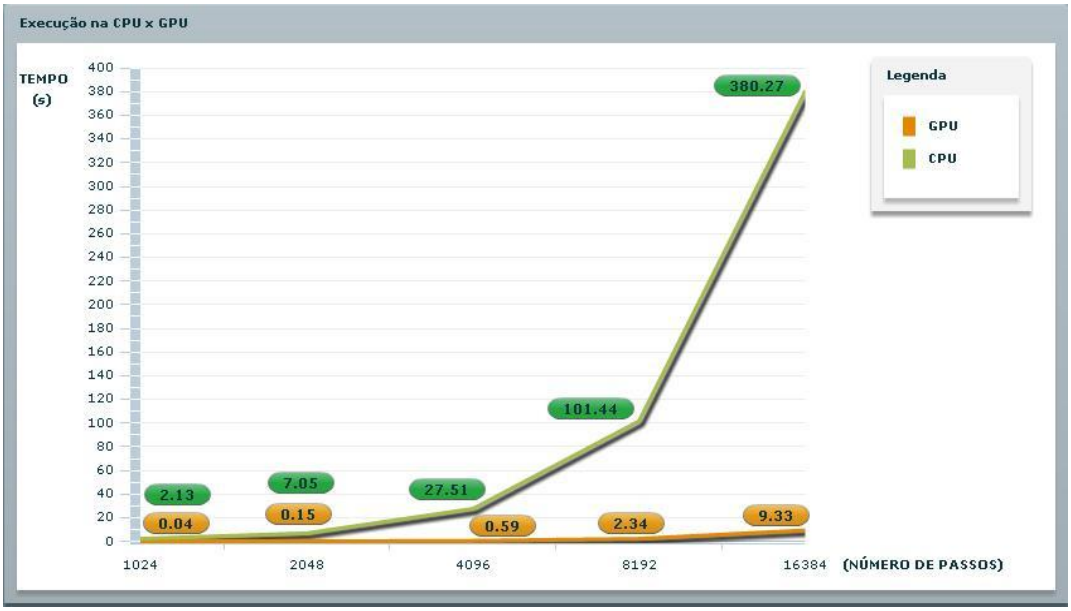

Gráfico 1 - Comparação do desempenho GPU x CPU.

Foram realizadas comparações de desempenho com outra ferramenta que fornece suporte à paralelização de aplicações,a biblioteca OpenMP,disponível para a linguagem C,

Foi utilizado o OpenMP em conjunto com o compilador GCC, por este incorporar os mecanismos de paralelização desta biblioteca. Durante o processo de compilação com o GCC foi necessário a inclusão do parâmetro -O para otimização do código durante o processo. A ausência desse parâmetro torna a aplicação menos eficiente sendo sensível a perda de desempenho durante os testes.

Nos testes foram utilizados 2 processadores da Intel modelos Core 2 Duo E8400 e Core 2 Quad Q9650 na paralelização da resolução de árvores trinomiais com a biblioteca OpenMP. A escolha deste processador justifica-se devido a similiaridade de preço com o conjunto Intel Core 2 Duo E8400 juntamente com a placa de vídeo NVidia® Geforce 8800 GTS, permitindo uma comparação justa, já que os custos de aquisição são compatíveis. Os resultados foram comparados aos obtidos com a aplicação CUDA™ rodando na GPU e podem ser observados no **Gráfico 2**.

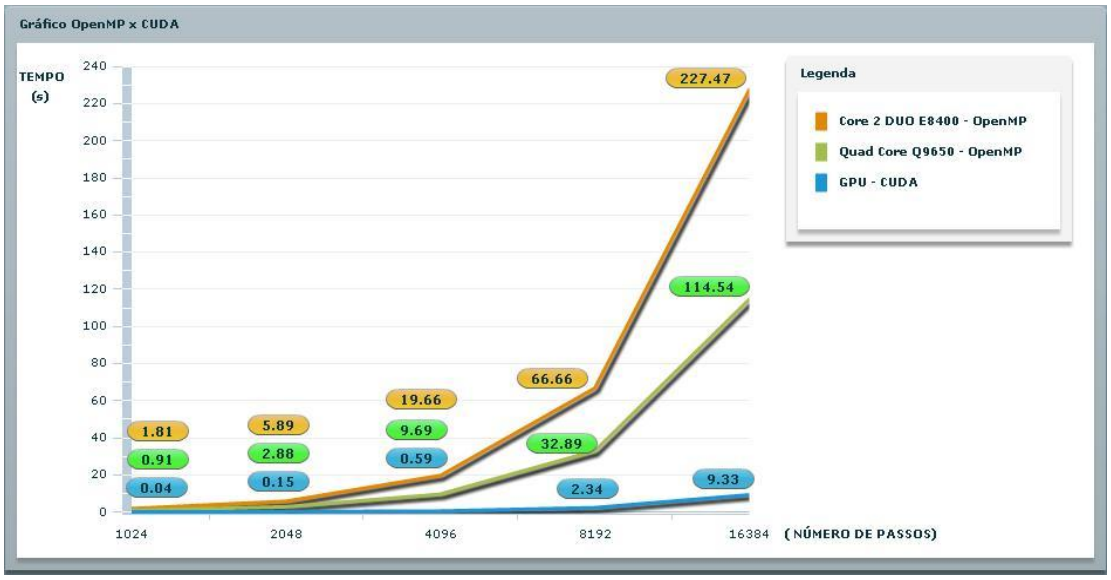

Gráfico 2 - Comparação de desempenho CUDA™ x OpenMP.

Na **Tabela 1** temos os valores obtidos com os testes e também o fator de ganho para as execuções dos sistemas citados.

Durante os testes da execução do programa CUDA™ na GPU realizou-se a medição do percentual de uso da CPU. Observou-se que aproximadamente 50% do processador E8400 estava sendo utilizado durante toda a execução do programa na GPU, ou seja, um dos núcleos era utilizado em 100% nesse processo. Ocorre que o processador permanece todo o tempo da execução na GPU realizando verificações quanto a conclusão da execução do kernel.

Tabela 1 - Tempos das execuções com CUDA™ x OpenMP.

Número de opções	Número de passos	Tempo OpenMP CPU E8400	Tempo OpenMP CPU Q9650	Tempo CUDA GPU 8800 GTS	Ganho GPU x E8400	Ganho GPU x Q9650
512	1024	1,812	0,906	0,038	47,684	23,842
	2048	5,890	2,875	0,149	39,530	19,295
	4096	19,656	9,688	0,589	33,372	16,448
	8192	66,656	32,891	2,340	28,485	14,056
	16384	227,467	114,540	9,330	24,380	12,277

Analisando os resultados obtidos nos testes foi possível constatar que a execução do programa na GPU mostrou desempenho muito superior se comparado ao algoritmo executando na CPU. Podem-se observar ganhos de tempo superiores a 40 vezes, até mesmo se comparado ao desempenho com aplicações que utilizam outros métodos de paralelismo como o OpenMP.

5. Conclusões e Trabalhos Futuros

Para o desenvolvimento deste trabalho foram estudados os métodos de precificação de opções, árvores trinomiais e a API CUDA™, onde foram demonstradas as vantagens da utilização de unidades de processamento gráfico (GPUs) no processamento de aplicações genéricas.

Foram executados testes e comparações de desempenho em todas as etapas dos processos de desenvolvimento e validação dos resultados obtidos com a aplicação.

A programação de aplicações genéricas em GPU permite a utilização de várias técnicas que

auxiliam enormemente nos processos de otimização de algoritmos. A utilização de processadores gráficos em aplicações altamente paralelizáveis como meio de maximização de recursos existentes ou até mesmo substituição de outras opções mostrou ser eficiente.

Desta forma, conclui-se que a utilização de unidades de processamento gráfico (GPU) é uma alternativa viável para se trabalhar com computação de alto desempenho nas diversas áreas de pesquisa científica.

Para trabalhos futuros, é possível citar o desenvolvimento de uma versão para GPU mais avançada do modelo de precificação de opções com utilização de árvores trinomiais que permitirá calcular e analisar riscos de investimento em tempo real sobre uma carteira de ações que considere exercício em qualquer período e o pagamento de dividendos. Outra possibilidade seria a implementação do modelo de precificação utilizado nesse trabalho com o emprego de múltiplas GPU's. Outro extensão a ser desenvolvida seria a implementação de uma versão da técnica proposta que faça uso do poder computacional da CPU e da GPU em conjunto.

Com o objetivo de se conseguir uma maior visualização das operações fundamentais ligadas aos mecanismos de aceleração do desempenho na execução dos sistemas na GPU, a implementação deverá ser avaliada com ferramentas que realizam um *profiling* da aplicação. O *profiling* permite a visualização da largura de banda de memória em um *kernel*. Com a utilização de uma ferramenta chamada CUDA Visual Profiler, é possível realizar o *profiling* de aplicações C rodando em GPU's.

Referências

(FARBER, 2008) FARBER, Rob – CUDA, Supercomputing for the Masses: Part 5. Disponível em: http://www.ddj.com/hpc-high-performance-computing/208801731>. Acessado em 30 de maio 2009.

(GROSSO, 2006) GROSSO, Luciano Monter de Pinho. Apreçamento de Opções sobre Futuro de Depósitos Inter-financeiros de um Dia. 2006. 84 f. Dissertação (Mestrado) – PUC-Rio, 2006.

(HARISH, 2007) HARISH, Pawab; NARAYANAN, P. J. Accelerating Large Graph Algorithms on the GPU Using CUDA. Center for Visual Information Technology International Institute of Information Technology Hyderabad, India, 2007.

(HULL, 2005) HULL, John C. – Fundamentos dos Mercados Futuros e de Opções. 4 ed., BM&F, Bolsa de Mercados e Futuros, 2005.

(LEMGRUBER, 1995) LEMGRUBER, Eduardo Facó – Avaliação de Contratos de Opções. BM&F – Bolsa de Mercados e Futuros, 1995.

(MORITSCH, 2006) MORITSCH, Hans. High Performance Computing in Finance - On the Parallel Implementation of Pricing and Optimization Models. 2006. 132 f. Dissertação (Doutorado) - Institut fur Softwaretechnik und Interaktive Systeme, 2006.

(SOUZA, 1999) SOUZA, Luiz Alvares Rezende de. Valor em Risco em Épocas de Crise. 1999. 122 f. Dissertação (Mestrado) – Universidade de São Paulo, FEA/USP, 1999.

(SBC, 2006) SBC. Grandes Desafios da Pesquisa em Computação no Brasil: 2006 - 2016. Desenvolvido pela Sociedade Brasileira de Computação. 2006. Disponível em <http:// 143.54.83.4/ ArquivosComunicacao/ Desafios_portugues.pdf> Acessado em: 30 de ago. 2008.

(ZONE, 2009) ZONE, CUDA. O que é CUDA. Disponível em: <http://www.nvidia.com.br/object/cuda_what_is_br.html>. Acessado em: 12 de maio de 2009.