Aumento da Eficiência Energética do Método DFT Através da Redução do Tempo de Cálculo Utilizando GPU

C.P. Silva[1], **L.F. Cupertino**[1], **D.S. Chevitarese**[1], **M.A.C. Pacheco** [1], **C. Bentes**[2]

[1]Departamento de Engenharia Elétrica
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) – Brasil

[2]Departamento de Engenharia de Sistemas e Computação
Universidade do Estado do Rio de Janeiro (UERJ) – Brasil

{cleomar,cuper,daniel,marco}@ele.puc-rio.br, cris@eng.uerj.br

Abstract. *Density functional theory (DFT) is one of the most popular and versatile methods available in condensed-matter physics, computational physics, and computational chemistry. It is the basis for most material simulation systems. However, due to the complexity of the calculations involved, DFT brings a huge demand for computing power. The large amount of time taken to do such computation increases energy requirements, which is nowadays one of the biggest concerns from the environmental perspective. In this work, we propose an energy-aware and efficient solution to DFT that takes benefit from the arithmetic capability of modern graphic cards (or GPUs). Our GPU implementation achieved significant acceleration over a standard CPU implementation, spending 20 times less energy.*

Resumo. *A Teoria do Funcional da Densidade (DFT) é um dos métodos mais populares e versáteis disponíveis na Física de Matéria Condensada, e na Física e Química Computacionais. É a base da maioria dos sistemas de simulação de materiais. Entretanto, devido à complexidade dos cálculos envolvidos, a DFT gera uma alta demanda de poder computacional. A grande quantidade de tempo necessária para realizar as computações envolvidas aumenta as necessidades de energia, o que, hoje em dia, constitui uma das maiores preocupações sob o ponto de vista ambiental. Neste trabalho, propomos uma solução eficiente e consciente do uso de energia que aproveita a grande capacidade aritmética das placas gráficas (ou GPUs) modernas. Nossa implementação em GPU alcançou uma aceleração significativa sobre uma implementação CPU tradicional, gastando 20 vezes menos energia.*

1. Introdução

A Teoria do Funcional da Densidade (DFT – *Density Functional Theory*) é um dos métodos iterativos mais usados para encontrar uma solução aproximada para a equação de Schrödinger a fim de estudar a estrutura eletrônica de sistemas de múltiplos corpos. O uso de DFT é bastante difundido na investigação das superfícies de catalisadores [Raybaud et al. 2000, Jacobsen et al. 2002, Yang et al. 2010] e no estudo de novos materiais na escala nanométrica, como nanotubos de carbono [Moradian et al. 2008] e grafeno [Qin et al. 2010].

Contudo, os cálculos realizados em DFT são computacionalmente intensos. Isso se deve ao cálculo das integrais de troca e correlação eletrônica, da energia de Hartree e energia cinética dos elétrons, as quais requerem maior esforço computacional à medida que o número de elétrons presentes na simulação aumenta. Grande parte dos experimentos realizados por DFT levam semanas ou até mesmo meses para serem realizados. Esse tempo de computação gera um impacto ambiental significativo, posto que os experimentos são executados em máquinas de grande porte, as quais possuem um alto consumo energético.

Com a crescente preocupação em reduzir as emissões de gases causadores do efeito estufa e o grande foco em questões de meio-ambiente e mudança climática, o uso de equipamentos com alta eficiência energética está em ascendência, tanto do ponto de vista tecnológico como do ponto de vista social. Na área da computação, a "computação verde" é a nova fronteira de desenvolvimento de hardware e de software. O foco que durante muito tempo foi o desenvolvimento de sistemas eficientes e com baixo custo, está agora também no consumo de energia e necessidade de refrigeração destes sistemas. Muitos esforços têm sido feitos neste sentido, na construção de sistemas operacionais [Vahdat et al. 2000], sistemas de memória [Reinman et al. 2002], sistemas de rede [Jiao and Hurson 2007], processadores [Chakraborty et al. 2007] e no desenvolvimento de novos algoritmos e aplicações [Rivoire et al. 2007, Kansal and Zhao 2008].

Portanto, a solução para DFT deve levar em consideração não somente a eficiência do algoritmo, mas também o consumo de energia gerado. É necessário que se maximize a eficiência para evitar longos tempos de computação, em um hardware que não precise de grande consumo de energia para tal. Neste cenário, propomos o uso de placas gráficas ou GPUs (*Graphics Processing Units*) para acelerar a computação do DFT.

GPUs são processadores de alto desempenho que foram inicialmente projetados para acelerar operações gráficas, mas hoje em dia têm sido largamente empregados na aceleração de aplicações científicas de propósito geral [Trapnell and Schatz 2009, Yang et al. 2007, Sun and Ma 2009]. Sua capacidade computacional provém de uma arquitetura altamente paralela, apresentando desempenho uma ordem de grandeza acima do desempenho de CPUs tradicionais para aplicações que requerem um elevado número de operações aritméticas sobre um grande volume de dados. Do ponto de vista do consumo de energia, entretanto, a transferência da carga computacional para um co-processador pode levar ao aumento do consumo total de energia. A GPU é um componente de hardware sofisticado com inúmeros elementos de processamento e requer uma quantiddade significativa de energia para funcionar.

Boa parte da literatura em implementações de aplicações científicas em GPUs, entretanto, tem focado somente na aceleração obtida pela GPU. Muito pouco tem sido estudado com relação ao consumo de energia gerado. Apenas alguns poucos trabalhos podem ser citados. Rofouei *et al.* [Rofouei et al. 2008] mediram com o *LEAP-server* o consumo de energia de aplicações em GPU, utilizando os exemplos provindos do SDK da NVIDIA. Sheaffer *et al.* [Sheaffer et al. 2005] exploraram técnicas de controle térmico para arquiteturas de placas gráficas. Ramani *et al.* [Ramani et al. 2006] apresentaram um *framework* para estimar o consumo de energia em GPUs. Takizawa *et al.* [Takizawa et al. 2008] propuseram um sistema adaptativo que escolhe o melhor processador em tempo de execução num sistema CPU-GPU para economizar energia. Huang *et al.* [Huang et al. 2009] apre-

sentaram um estudo de características de desempenho, potência e energia do software GEM para cálculo do potencial eletrostático dentro de uma molécula.

Neste trabalho, estudamos não somente a aceleração obtida pelo uso da GPU na resolução do Hamiltoniano de DFT, mas também caracterizamos o consumo de energia gerado pela solução proposta. A resolução do Hamiltoniano de DFT engloba cinco termos: a energia cinética, as contribuições locais e não-locais do pseudopotencial, o potencial de Hartree e o potencial de troca e correlação. Como alguns desses termos foram implementados anteriormente em GPU (vide [Yasuda 2008, Genovese et al. 2009]), propomos aqui a implementação em GPU de um termo do Hamiltoniano que ainda não foi explorado: o potencial de Hartree. Para avaliar os ganhos e gastos, utilizamos uma aplicação real de química quântica: determinação da energia do estado fundamental de um nanotubo de carbono. A fim de mensurar a eficiência energética da implementação em GPU, focamos no equacionamento de Poisson, gerando vários possíveis casos com entradas de tamanhos variados. Os resultados demonstram que a implementação desta aplicação em GPU, possibilita, não somente uma diminuição no tempo computacional, como também no consumo energético. Nos experimentos deste artigo, a redução de consumo foi de até 20 vezes.

Este trabalho está organizado da seguinte maneira. Na seção 2, descrevemos DFT brevemente. Na seção 3, mostramos os fundamentos de programação em placa gráfica. A seção 4 mostra como a DFT foi implementada em GPU. Por fim, as seções 5 e 6 apresentam os resultados e conclusões.

2. Teoria do Funcional da Densidade

A equação de Schrödinger [Schrödinger 1926] marcou o início da mecânica quântica moderna. Esta equação determina a função de onda quântica de um sistema (um átomo, uma molécula ou um sólido), contendo toda a informação necessária para determinar o estado do mesmo.

$$H\Phi = E\Phi \tag{1}$$

onde H é o Hamiltoniano, E é a energia e Φ é a função de onda. Entretanto, poucos sistemas físicos possuem soluções analíticas e, muitas vezes, a solução numérica pode ser computacionalmente inviável, devido ao tamanho do problema.

A Teoria da Densidade Funcional é um dos métodos mais populares e versáteis disponíveis para estudos de química computacional e da física do estado sólido. DFT é um método iterativo para encontrar uma solução aproximada à equação de Schrödinger. Ele é usado para estudar a estrutura eletrônica (principalmente estado fundamental) de sistemas de muitos corpos, em particular os átomos, moléculas e sólidos. Seu formalismo foi estabelecido a partir dos dois teoremas de Hohenberg e Kohn [Hohenberg and Kohn 1964]. Eles mostraram que, em princípio, a densidade de elétrons contém todas as informações que podem ser obtidas a partir da função de onda de muitos elétrons.

Algumas aproximações são usadas para encontrar uma solução para a equação de Schrödinger. Uma delas é a abordagem de Born-Oppenheimer, a qual desacopla o movimento dos núcleos e dos elétrons, considerando que o movimento dos elétrons ocorrem em um campo nuclear fixo [Born and Oppenheimer 1927]. Para os termos de troca e correlação, as abordagens mais conhecidas são a aproximação de densidade local (LDA

– *Local-Density Approximation*) e a aproximação do gradiente generalizado (GGA – *Generalized Gradient Approximation*). Estas abordagens da Teoria do Funcional da Densidade nos permitem escrever o Hamiltoniano de Kohn-Sham como visto na equação (2) [Kohn et al. 1965].

$$H_{KS} = -\frac{1}{2}\nabla^2 + V^{PS}(r) + \int \frac{\rho(r)}{|r - r'|}dr + V_{xc}(r) \tag{2}$$

Usando o procedimento descrito por Kleinman e Bylander [Kleinman and Bylander 1982] para separar o potencial $V^{PS}(r)$ em componentes locais (longo alcance) e não locais (curto alcance), obtem-se a equação (3).

$$H_{KS} = -\frac{1}{2}\nabla^2 + V^{PS}_{ion,local}(r) + V^{KB}_{nao-local,l}(r) + \int \frac{\rho(r)}{|r - r'|}dr + V_{xc}(r) \tag{3}$$

Um dos programas mais utilizados para o cálculo do Hamiltoniano (equação (2)) é o SIESTA. Este concentra seu custo computacional na solução da equação de Poisson, sendo responsável por até 60% do tempo de processamento. O detalhamento de ambos se encontra a seguir.

2.1. SIESTA

O SIESTA (*Spanish Initiative for Electronic Simulations with Thousands of Atoms*) é um sistema auto-consistente da Teoria do Funcional da Densidade, que usa a representação pseudopotencial do núcleo iônico, orbitais atômicos numéricos e conjunto de bases localizadas. Esse sistema representa um esforço para se desenvolver um método DFT de *ab initio* de ordem N auto-consistente. A determinação das operações do Hamiltoniano auto-consistente em $O(N)$ é difícil de ser atingida usando ondas planas como conjunto de base. Isto levou a escolha de um conjunto de bases atômicas localizadas para o SIESTA [Soler et al. 2002, Ordejón et al. 1996]. No entanto, algumas funções ainda necessitam de um maior número de operações. No método do SIESTA, as equações (2) e (3) são reescritas da seguinte forma (4).

$$H = T_s + \sum_I V_I^{local} + \sum_I V_I^{KB} + V_H(r) + V_{xc}(r) \tag{4}$$

onde I representa o índice atômico, $\sum_I V_I^{local}$ e $\sum_I V_I^{KB}$ são, respectivamente, as contribuições local e não local do pseudopotencial, T_s é a energia cinética, $V_H(r)$ é o potencial de Hartree e $V_{xc}(r)$ é o potencial de troca e correlação.

O termo do pseudopotencial local é um operador de longo alcance que pode ser dispendioso em termos computacionais. Assim, o método utiliza um potencial V_I^{NA} para substituir o potencial V_I^{local}, o qual é definido por $V_I^{NA} = V_I^{local} + V_I^{atomico}$ [Soler et al. 2002, Ordejón et al. 1996]. O termo $V_I^{atomico}$ é o potencial criado pela distribuição de elétrons com densidade eletrônica $\rho^{atomico}$.

$$H = T_s + \sum_I V_I^{KB} + \sum_I (V_I^{NA}(r) - V_I^{atomico}(r)) + V_H(r) + V_{xc}(r) \tag{5}$$

O potencial de Hartree $V_H(r)$ pode ser reescrito como a equação (6).

$$V_H(r) = \delta V_H(r) + \sum_I (V_I^{atomico}(r)) \tag{6}$$

Assim, o Hamiltoniano utilizado no SIESTA assume sua forma final (equação (7)).

$$H = T_s + \sum_I V_I^{KB} + \sum_I V_I^{NA}(r) + \delta V_H(r) + V_{xc}(r) \tag{7}$$

2.2. Equação de Poisson

A equação de Poisson é uma equação diferencial parcial amplamente utilizada na engenharia e na física teórica. No espaço euclidiano esta equação é freqüentemente escrita como:

$$\nabla^2 \phi = f \tag{8}$$

O quarto termo do lado direito da equação (7), o potencial de Hartree, $\delta V_H(r)$, é produzido pela diferença da densidade atômica e auto-consistente $\delta\rho(r)$. A equação de Poisson usada no SIESTA é baseada em condições periódicas de contorno, podendo, portanto, ser resolvida no domínio da freqüência. A solução dessa equação é usada para encontrar o potencial de Hartree associada com a variação de densidade.

3. Programação em GPU

A grande necessidade de poder computacional e a demanda por gráficos em alta definição da indústria de jogos e entretenimento digitais, motivou os poderosos avanços obtido pelas GPUs nos últimos anos. O aumento do poder de processamento das placas gráficas ocorreu em taxas muito maiores do que os processadores convencionais.

A principal diferença entre uma GPU e uma CPU tradicional está no espaço gasto no chip com processamento. A GPU utiliza muito mais transistores para processamento do que para controle de fluxo ou memória de acesso rápido, conforme mostra a Figura 1. Mais especificamente, a GPU é especialmente concebida para lidar com problemas que podem ser tratados sob a forma de computações de dados em paralelo de grande intensidade aritmética. Dado que um mesmo programa é executado recorrentemente sobre cada elemento de dados, não existe a necessidade de um sofisticado fluxo de controle.

Figura 1. Diferença no espaço gasto no chip de uma CPU e uma GPU

3.1. Hardware da GPU

Uma GPU consiste em um conjunto de multiprocessadores (*Streaming Multiprocessors* – SM), conforme pode ser observado na Figura 2. O programa que executa na GPU deve ser dividido em uma grade composta de blocos de *threads*. Os blocos são enumerados e distribuídos para os multiprocessadores disponíveis. As *threads* de um bloco são executadas, de forma concorrente, em um multiprocessador, e quando esse bloco é finalizado, outro bloco é inicializado até que toda a grade seja finalizada.

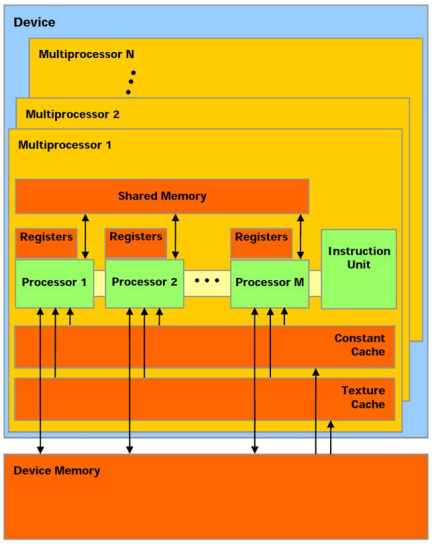

Figura 2. Arquitetura de uma GPU

Os multiprocessadores possuem até oito núcleos chamados Processadores Escalares (*Scalar Processors* – SP), duas unidades para números transcendentais (ex: π, e, etc.), uma unidade para controle das *threads* e a memória compartilhada. Os multiprocessadores criam, gerenciam e executam as *threads* em hardware sem custo. Além disso, a barreira de sincronismo entre as *threads* é implementada em uma única instrução e resolvida de maneira eficiente.

Em termos de hierarquia de memória, cada multiprocessador possui até 16384 registradores de 32 bits de uso exclusivo (2048 registradores por processador, totalizando 8Kb), uma memória compartilhada (*shared memory*) com 16Kb de tamanho e acesso ultra-rápido, acessível por todas as *threads* de um bloco, além de duas memórias somente-leitura de rápido acesso: a memória de textura e a memória constante. Adicionalmente, é possível acessar a memória principal da GPU (*device memory*), onde reside a memória local de cada *thread* e a memória global da aplicação. É a memória com a maior latência de acesso.

Para conseguir gerenciar centenas ou milhares de *threads* ao mesmo tempo, os multiprocessadores empregam uma arquitetura chamada SIMT (Uma Instrução Múltiplas Threads – *Single-Instruction Multiple-Thread*). Nela cada *thread* é alocada a uma SP, que a executa independentemente. A unidade SIMT do multiprocessador cria, gerencia, agenda e executa as *threads* em grupos chamados *warps*. Os integrantes desses grupos são inicializados ao mesmo tempo, mas possuem liberdade para parar ou executar de forma independente.

3.2. Programando com CUDA

A tecnologia CUDA, criada em 2006 pela NVIDIA, oferece um ambiente que permite o desenvolvimento em linguagem C, C++, FORTRAN, OpenCL e DirectX Compute. A arquitetura da NVIDIA possui três níveis de abstração: a hierarquia de grupos de threads, as memórias compartilhadas e as barreiras de sincronização.

Na arquitetura CUDA as funções são chamadas *kernels* e são invocadas, no caso da programação em C, por extensões da linguagem. Cada *thread* possui um identificador único permitindo que o desenvolvedor acesse cada uma delas. As *threads* são hierarquizadas em blocos tridimensionais e esses são organizados em grades de duas dimensões, segundo mostra a Figura 3. Essa hierarquia torna mais fácil o entendimento ao se computar elementos vetoriais e matriciais, uma vez que os índices das threads são sequenciais dentro dos blocos.

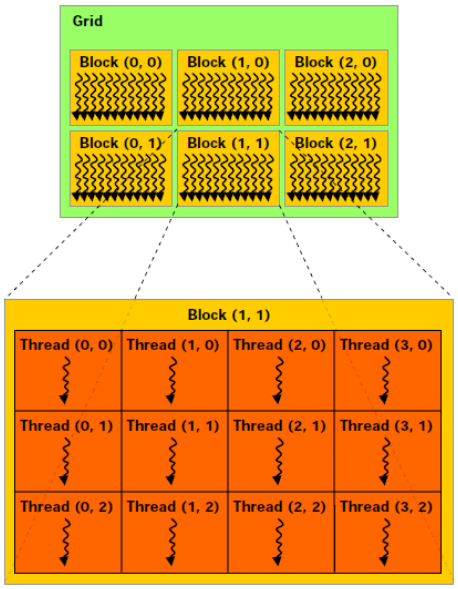

Figura 3. Grade de blocos de threads

As *threads* de um mesmo bloco podem cooperar entre si através do sincronismo de suas execuções e da memória compartilhada. A única restrição ao uso do bloco de *threads* e sua memória compartilhada é seu tamanho limitado. Entretanto, é possível dividir todo o *kernel* em blocos iguais e organizá-los numa grade para que o máximo possível deles seja executado ao mesmo tempo pela placa. Nesse caso, o número de *threads* de uma função a ser executada será o número de *threads* por bloco vezes o número de blocos. A memória global é visível por todo o programa, não apenas dentro dos blocos.

4. Teoria do Funcional da Densidade em GPU

A última versão estável do código fonte do SIESTA, originalmente escrito em Fortran, foi utilizada para nossa implementação em GPU. No SIESTA, o termo do potencial de Hartree, $\delta V_H(r)$, é encontrado pela solução da equação de Poisson, a qual é baseada em condições periódicas de contorno e é resolvida no domínio da freqüência. Inicialmente, são encontrados os componentes da Transformada de Fourier da densidade eletrônica $\delta\rho(r)$. De forma combinada, na mesma seção do código, o somatório da contribuição

de cada elemento da matriz de densidades é usado para encontrar a energia de Hartree e as forças de deslocamento atômicas. O potencial de Hartree é calculado e em seguida transformado para o espaço real conforme o ilustrado na Figura 4.

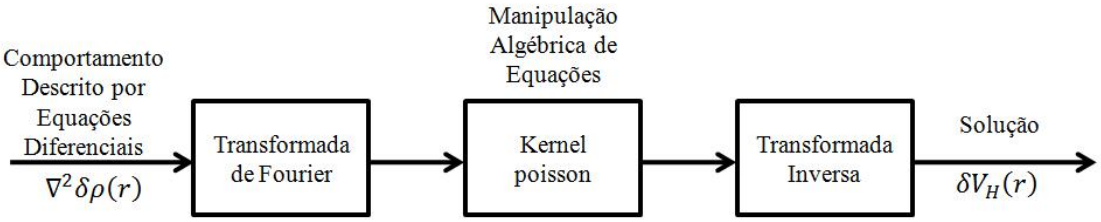

Figura 4. Solução do termo do potencial de Hartree

A Transformada de Fourier originalmente empregada no código SIESTA é o algoritmo GPFA (*Generalized Prime Factor FFT*) [Temperton 1992]. Na equação de Poisson implementada em GPU, a transformada foi substituída pelo algoritmo CUFFT (*CUDA Fast Fourier Transform*) [NVIDIA 2007]. Assim, a Transformada de Fourier da matriz de densidades $\delta\rho(r)$ e a Transformada Inversa de Fourier do potencial de Hartree, $\delta V_H(r)$, são realizadas na placa gráfica pelo CUFFT.

Para o cálculo da equação de Poisson em GPU, desenvolvemos dois *kernels* diferentes. O primeiro *kernel* computa o potencial de Hartree para os elementos da matriz de densidade e aproveita que a informação da matriz de densidades foi lida da memória global para calcular as contribuições de cada elemento para a energia de Hartree e para as forças atômicas. Uma vez que o sincronismo entre blocos distintos só pode ser feito por *kernels* distintos, implementamos um segundo *kernel* para finalizar as somas necessárias para energia de Hartree e forças atômicas.

O primeiro *kernel* foi projetado de tal forma que cada *thread* processa um vetor de dados da matriz de densidades $\delta\rho(r)$, conforme mostra a Figura 5. Este *kernel* realiza pré-somatórios das contribuições dos elementos da matriz de densidades $\delta\rho(r)$ para os cálculos da energia de Hartree (um somatório) e das forças de deslocamento atômicas (seis somatórios). Os pré-somatórios são realizados para cada bloco de *threads*, reduzindo de 1792 para 7 o número de escritas necessária na memória global para cada bloco. Cada bloco com 256 *threads* utiliza 28 registradores e 7200 bytes de memória compartilhada.

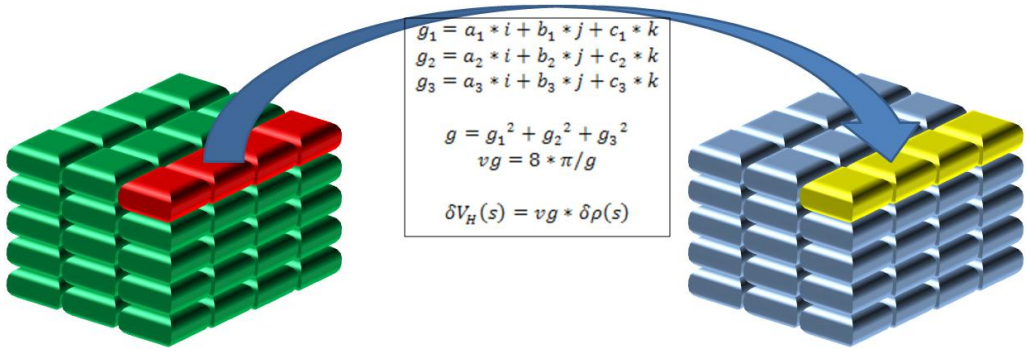

Figura 5. Cada *thread* processa um vetor de dados da matriz de densidades

O segundo *kernel* é responsável pela obtenção dos valores finais. Este utiliza 13

registradores e 3620 bytes de memória compartilhada para cada bloco com 256 *threads*.

Para os somatórios realizados nos dois *kernels*, reduzimos o número de operações de adição realizadas com *warps* incompletos. Para tal, calculamos a norma-1 dos sete vetores em paralelo, seguindo o padrão ilustrado na Figura 6. Neste caso, ao invés de declarar sete espaços na memória compartilhada, um único espaço de memória é declarado. Os elementos pertencentes a um mesmo vetor são armazenados mantendo-se um intervalo de seis posições. Essas posições intermediárias correspondem aos seis demais vetores.

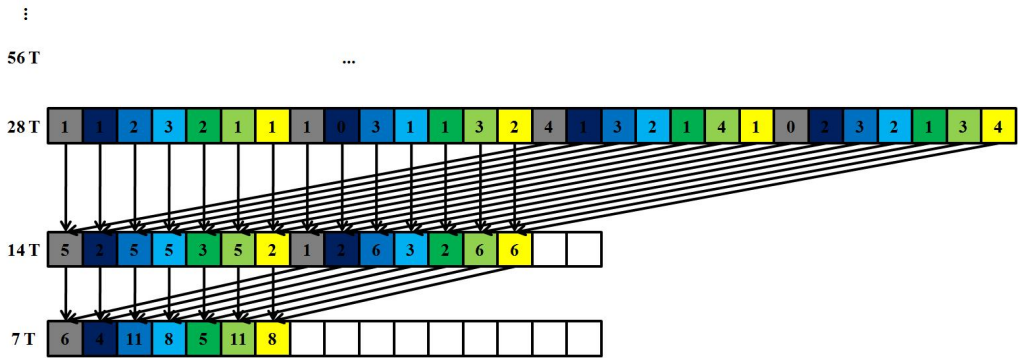

Figura 6. Somatório paralelo de sete vetores na memória compartilhada da GPU

O somatório paralelo dos sete vetores, da forma como ilustrado na Figura 6, realiza um padrão de acesso linear na memória compartilhada da GPU e, portanto, não há conflitos de bancos. Na etapa anterior ao somatório paralelo, as *threads* que calculam os valores a serem somados, acessam a memória compartilhada segundo um padrão de seis intervalos entre cada elemento acessado, o qual consiste em outro padrão de acesso livre de conflitos de bancos.

5. Resultados

Para avaliar o desempenho e o consumo de energia de nossa implementação em GPU do potencial de Hartree, utilizamos uma placa gráfica Tesla C1060. A Tesla C1060 possui 240 elementos de processamento (1.3 GHz) e 4 GB de memória RAM com largura de banda de acesso à memória de 102 GB/segundo através de um barramento de 512 bits. O computador utilizado possui quatro Teslas e um processador AMD Phenom II com 4 núcleos na freqüência de 3 GHz, com 16 GB de memória RAM. Todas as comparações foram realizadas entre uma GPU e um núcleo da CPU.

Para medir o consumo, o computador foi ligado a um medidor de potência da marca Yokogawa, modelo MCP5000, onde foi registrada a corrente máxima (A) e a energia total dissipada em kWh. A montagem seguiu o esquema da Figura 7. Nossos experimentos se baseiam em medidas reais, obtidas com o medidor de potência, da energia consumida. Este procedimento apresenta a vantagem de ser mais preciso quando comparado a outros procedimentos que levam em consideração o consumo médio, como em [Rofouei et al. 2008]. Contudo, por conta da resolução do medidor empregado, somente foi possível realizar medidas para matrizes de densidades com dimensões superiores a 25 milhões de elementos. Este tamanho mínimo encontra-se na ordem de grandeza dos problemas de interesse para aplicação de DFT acelerado por GPU.

O cálculo da eficiência energética é realizado segundo a equação (9), em que E representa a energia em kWh. Já o cálculo da aceleração é realizado segundo a equação (10), em que t representa o tempo de execução.

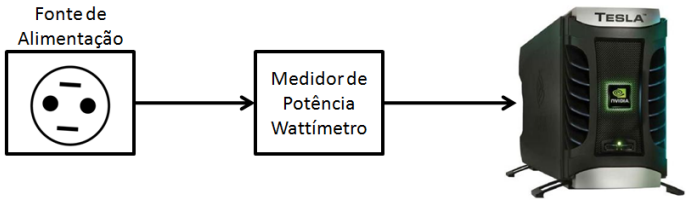

Figura 7. Configuração da medição de consumo dos experimentos

$$eficiencia = E_{CPU}/E_{GPU} \quad (9) \qquad aceleracao = t_{CPU}/t_{GPU} \quad (10)$$

Na avaliação de nossa implementação, realizamos dois experimentos distintos. No primeiro, comparamos somente a porção DFT que foi paralelizada para GPU, com a implementação da mesma porção no código sequencial do SIESTA. Neste caso, foram comparadas as partes do código que realizam o cálculo do potencial de Hartree, energia de Hartree e forças atômicas. A dimensão da matriz de entrada foi variada de 25 a 250 milhões de elementos, em intervalos de 50 milhões. Durante o cálculo do potencial de Hartree para as diferentes entradas, medimos o tempo gasto, o consumo energético e a corrente máxima dissipada. Com esses dados, foi possível calcular a eficiência energética e a aceleração obtidas.

O segundo experimento consiste de uma execução completa do algoritmo DFT, com o termo do potencial de Hartree implementado em GPU. Sendo assim, os resultados dessa implementação são relativos a um modelo híbrido (CPU–GPU). Utilizamos DFT no cálculo da energia do estado fundamental de um nanotubo de carbono com 46 átomos. O nanotubo estudado neste experimento possui uma matriz com 100 milhões de pontos. Para a execução de ambas implementações (CPU e CPU–GPU), todas as medições do primeiro experimento foram realizadas novamente.

O gráfico da Figura 8 mostra a energia gasta em kWh, para o primeiro experimento. A linha tracejada indica a execução em CPU e a contínua em GPU. Conforme podemos observar neste gráfico, a energia consumida aumentou proporcionalmente à entrada, tanto na implementação sequencial, quanto na implementação GPU. Isso pode ser evidenciado ao se realizar uma regressão linear dos dados das duas curvas, as quais atingem coeficientes de correlação muito próximos do máximo. A diferença entre o consumo total da CPU e GPU também aumenta conforme a entrada. Isso se deve principalmente ao menor tempo de execução do algoritmo na GPU.

Durante este experimento, observamos que ao executar o algoritmo na placa gráfica, a corrente máxima consumida atingiu 4,6A, enquanto que durante o algoritmo sequencial na CPU, a corrente se manteve na faixa de 3,5A. Os valores máximos da corrente para cada tamanho de entrada se encontram na Tabela 1. Conforme podemos observar na tabela, há um consumo instantâneo maior da GPU frente à CPU. Entretanto, a diminuição

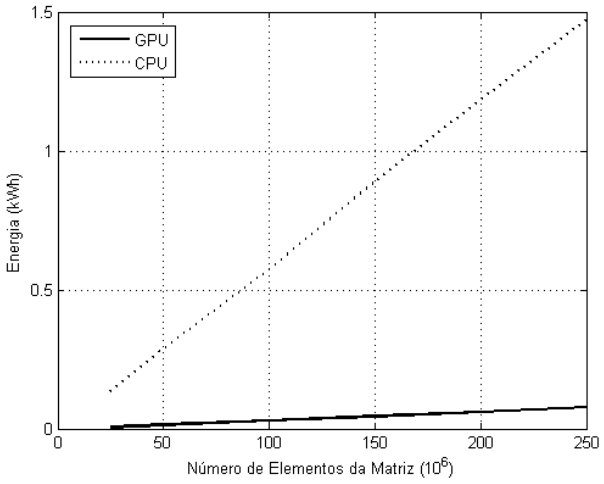

Figura 8. Energia por número de elementos da matriz

Entrada ($\times 10^6$)	CPU (A)	GPU (A)
25	3,324	4,332
50	3,522	4,505
100	3,479	4,634
150	3,481	4,660
200	3,329	4,606
250	3,259	4,625

Tabela 1. Corrente máxima (A) consumida para cada tamanho de entrada

no tempo de execução da aplicação, faz com que o consumo de energia total em kWh da GPU seja sempre bastante inferior ao da CPU.

O gráfico da Figura 9 mostra a eficiência energética e a aceleração obtidas para o primeiro experimento. A linha tracejada mostra a aceleração segundo a equação (10) e a linha contínua mostra a eficiência segundo a equação (9). Conforme podemos observar neste gráfico, a aceleração obtida se manteve praticamente constante, em torno de 26,8 vezes. Para a faixa de tamanhos de entrada utilizada neste estudo (que é a principal faixa de interesse prático), observamos pouca variação da aceleração com o tamanho do problema. Isto significa que todos os tamanhos de entrada utilizados são grandes o suficiente para manter a ocupação máxima da GPU. Para testes com sistemas físicos de apenas um átomo na célula unitária, com matriz de densidade menor, a aceleração obtida foi bem abaixo de 26,8 vezes. Porém, problemas pequenos não representam o interesse de utilização de DFT acelerado por GPU.

Com relação à eficiência energética, apesar de possuir um pico de corrente mais elevado, a implementação GPU apresentou uma eficiência energética com comportamento similar ao da aceleração. O maior valor de eficiência energética observada na faixa de realização do experimento foi de 22,33 e a eficiência energética média foi de 19,75. Observamos que o consumo de energia da GPU é proporcional ao tempo de execução, sendo o desempenho e a eficiência energética altamente sincronizados, tal qual observado por Huang *et al.* em [Huang et al. 2009].

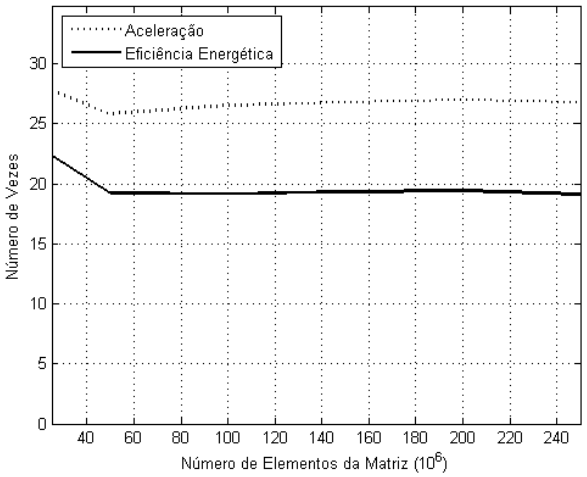

Figura 9. Aceleração e eficiência energética por número de elementos da matriz

No segundo experimento, avaliamos o desempenho e o ganho de energia da execução completa de DFT. Obtivemos ganhos de desempenho e consumo de energia quando comparado com a execução em CPU. A solução CPU–GPU se mostrou 3,3 vezes mais econômica em termos de energia e 3,7 vezes mais rápida que a solução CPU. Tal resultado indica que, mesmo que um código não esteja totalmente implementado em GPU, identificando-se seu gargalo, os ganhos de desempenho e energia compensam o trabalho de implementação em GPU.

6. Conclusões

Neste trabalho, apresentamos uma implementação em GPU da Teoria do Funcional da Densidade (DFT), com dois objetivos diferentes. O primeiro de avaliar a aceleração obtida pelo uso da GPU na resolução do potencial de Hartree. O segundo de caracterizar o consumo de energia gerado pela solução proposta, já que o consumo energético tem se tornado uma das principais preocupações atuais do ponto de vista ambiental.

Apesar da corrente drenada pela placa gráfica ser maior, o fato de ela trabalhar por um período menor de tempo que a CPU, possibilita uma economia de energia significativa. Nossos experimentos mostram que a implementação, mesmo que parcial de DFT em GPU, possibilita não só uma diminuição significativa no tempo de execução, como também no consumo energético. A redução de consumo obtida foi de até 20 vezes para o cálculo do potencial de Hartree e até 3,3 vezes para o cálculo completo de DFT.

Como trabalho futuro, pretendemos implementar em GPU os demais termos do Hamiltoniano, o que deverá trazer reduções ainda mais significativas do tempo total de cálculo do sistema SIESTA e da energia consumida por esta classe de problemas.

Referências

Born, M. and Oppenheimer, R. (1927). On the quantum theory of molecules. *Annalen der Physik*, 84:457.

Chakraborty, K., Wells, P. M., Sohi, G. S., and Chakraborty, K. (2007). A case for an over-provisioned multicore system: Energy efficient processing of multithreaded programs. Technical report.

Genovese, L., Ospici, M., Deutsch, T., Méhaut, J., Neelov, A., and Goedecker, S. (2009). Density Functional Theory calculation on many-cores hybrid CPU-GPU architectures. *arXiv*, 904.

Hohenberg, P. and Kohn, W. (1964). Inhomogeneous electron gas. *Phys. Rev*, 136(3B):B864–B871.

Huang, S., Xiao, S., and Feng, W. (2009). On the energy efficiency of graphics processing units for scientific computing. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8.

Jacobsen, C. J. H., Dahl, S., Boisen, A., Clausen, B. S., Topsøe, H., Logadottir, A., and Nørskov, J. K. (2002). Optimal catalyst curves: Connecting density functional theory calculations with industrial reactor design and catalyst selection. *Journal of Catalysis*, 205(2):382 – 387.

Jiao, Y. and Hurson, A. R. (2007). Energy-efficient wireless information retrieval. *J. Comput. Syst. Sci.*, 73(8):1145–1163.

Kansal, A. and Zhao, F. (2008). Fine-grained energy profiling for power-aware application design. *SIGMETRICS Perform. Eval. Rev.*, 36(2):26–31.

Kleinman, L. and Bylander, D. (1982). Efficacious form for model pseudopotentials. *Physical Review Letters*, 48(20):1425–1428.

Kohn, W., Sham, L., et al. (1965). Self-consistent equations including exchange and correlation effects. *Phys. Rev*, 140(4A):A1133–A1138.

Moradian, R., Behzad, S., and Azadi, S. (2008). Ab initio density functional theory investigation of electronic properties of semiconducting single-walled carbon nanotube bundles. *Physica E: Low-dimensional Systems and Nanostructures*, 40(10):3055 – 3059.

NVIDIA (2007). *CUDA CUFFT Library*. nVidia Corporation.

Ordejón, P., Artacho, E., and Soler, J. (1996). Self-consistent order-N density-functional calculations for very large systems. *Physical Review B*, 53(16):10441–10444.

Qin, W., Li, X., Bian, W.-W., Fan, X.-J., and Qi, J.-Y. (2010). Density functional theory calculations and molecular dynamics simulations of the adsorption of biomolecules on graphene surfaces. *Biomaterials*, 31(5):1007 – 1016.

Ramani, K., Ibrahim, A., and Shimizu, D. (2006). Powerred: A flexible modeling framework for power efficiency exploration in gpus. In *Proceedings of the Workshop on General Purpose Processing on GPUs, GPGPU'07*.

Raybaud, P., Hafner, J., Kresse, G., Kasztelan, S., and Toulhoat, H. (2000). Structure, energetics, and electronic properties of the surface of a promoted mos2 catalyst: An ab initio local density functional study. *Journal of Catalysis*, 190(1):128 – 143.

Reinman, G., Calder, B., and Austin, T. M. (2002). High performance and energy efficient serial prefetch architecture. In *ISHPC '02: Proceedings of the 4th International Symposium on High Performance Computing*, pages 146–159.

Rivoire, S., Shah, M. A., Ranganathan, P., and Kozyrakis, C. (2007). Joulesort: a balanced energy-efficiency benchmark. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 365–376.

Rofouei, M., Stathopoulos, T., Ryffel, S., Kaiser, W., and Sarrafzadeh, M. (2008). Energy-Aware High Performance Computing with Graphic Processing Units. In *Workshop on Power Aware Computing and System*.

Schrödinger, E. (1926). An undulatory theory of the mechanics of atoms and molecules. *Physical Review*, 28:1049–1070.

Sheaffer, J. W., Skadron, K., and Luebke, D. P. (2005). Studying thermal management for graphics-processor architectures. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, 2005*, pages 54–65.

Soler, J., Artacho, E., Gale, J., García, A., Junquera, J., Ordejón, P., and Sánchez-Portal, D. (2002). The SIESTA method for ab initio order-N materials simulation. *Journal of Physics: Condensed Matter*, 14:2745–2779.

Sun, W. and Ma, Z. (2009). Count Sort for GPU Computing. In *2009 15th International Conference on Parallel and Distributed Systems*, pages 919–924. IEEE.

Takizawa, H., Sato, K., and Kobayashi, H. (2008). Sprat: Runtime processor selection for energy-aware computing. In *IEEE Cluster*, pages 386–393. IEEE Computer Society.

Temperton, C. (1992). A Generalized Prime Factor FFT Algorithm for any N=2p3q5r. *SIAM Journal on Scientific and Statistical Computing*, 13:676.

Trapnell, C. and Schatz, M. (2009). Optimizing data intensive GPGPU computations for DNA sequence alignment. *Parallel Computing*.

Vahdat, A., Lebeck, A., and Ellis, C. S. (2000). Every joule is precious: the case for revisiting operating system design for energy efficiency. In *EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 31–36.

Yang, J., Wang, Y., and Chen, Y. (2007). GPU accelerated molecular dynamics simulation of thermal conductivities. *Journal of Computational Physics*, 221(2):799–804.

Yang, S., Adjaye, J., McCaffrey, W. C., and Nelson, A. E. (2010). Density-functional theory (dft) study of arsenic poisoning of nimos. *Journal of Molecular Catalysis A: Chemical*, 321(1-2):83 – 91.

Yasuda, K. (2008). Accelerating density functional calculations with graphics processing unit. *Journal of Chemical Theory and Computation*, 4(8):1230–1236.