Cálculos de Química Quântica em GPUs: proposta de um algoritmo paralelo para a pseudodiagonalização de matrizes simétricas usando a plataforma NVIDIA/CUDA.

Júlio Daniel de Carvalho Maia[1], Lucídio dos Anjos Formiga Cabral[1] e Gerd Bruno Rocha[2]

[1]Centro de Informática – Campus I – Universidade Federal da Paraíba (UFPB)
CEP – 58051-900 – João Pessoa – PB – Brasil

[2]Departamento de Química – Campus I – Universidade Federal da Paraíba (UFPB)
CEP – 58051-900 – João Pessoa – PB – Brasil

juliodaniel@di.ufpb.br, lucidio@ci.ufpb.br e gbr@quimica.ufpb.br

http://www.quantum-chem.pro.br

Abstract. *Eigenvalue and eigenvectors notions are essential for the calculations in Quantum Chemistry, since molecules and atoms energy levels are calculated based on the eigenvectors of some matrices. The diagonalisation procedure is a crucial step of the Hartree-Fock-Roothaan method, since it provides the molecular orbitals energies. However, finding an accurate set of eigenvalues and eigenvectors is not always mandatory. This paper focuses on a parallel implementation for an approximated eigenvector calculation algorithm on GPUs using the NVIDIA/CUDA platform.*

Resumo. *Noções de autovalores e autovetores de transformações lineares são essenciais em cálculos de química quântica, pois os níveis de energia dos átomos e moléculas são dados por autovalores de determinadas matrizes. O procedimento de diagonalização de uma matriz é um passo crucial no método de Hartree-Fock-Roothaan, visto que o mesmo dará as energias dos orbitais moleculares.Porém, nem sempre é necessário encontrar um conjunto solução exato. Nesse artigo, mostramos a implementação de um algoritmo paralelo para GPUs que usa aproximações para o cálculo de autovetores proposto por [Stewart,1981], utilizando a plataforma NVIDIA/CUDA.*

1. Introdução

A possibilidade de realizar cálculos matemáticos de alta demanda com as GPUs está revolucionando diversas áreas do conhecimento onde a dependência computacional é crítica (Hwu, 2011). E é dessa forma que a química tem se aproveitado dessa nova possibilidade, pois, atualmente existe uma crescente necessidade de modelar sistemas de elevada complexidade molecular(Rocha & Simas, 2007) e (Maia et al., 2012).

É com esse propósito que apresentamos nesse manuscrito a proposta de um algoritmo paralelo para a pseudodiagonalização de uma matriz simétrica que possa se aproveitar do elevado paralelismo das GPUs.

Uma estratégia que seguimos é a substituição de etapas que são computacionalmente custosas e que possuem algoritmos de elevada complexidade por métodos e algoritmos numéricos mais eficientes e de menor complexidade computacional. Podemos citar como exemplo a diagonalização de matrizes simétricas, $O(n^3)$.

Para a resolução das equações dos métodos quânticos faz-se necessário o emprego de um procedimento numérico iterativo conhecido como *Self-Consistent Field* (SCF), onde partindo-se de uma geometria molecular fixa podemos encontrar a energia total desse sistema, bem como a matriz densidade, que é a solução das equações. Um dos passos do SCF é justamente a diagonalização da matriz de Fock, **F**.

A abordagem de Stewart e colaboradores (Stewart, Császár, & Pulay, 1982)segue a lógica de que o procedimento de diagonalização não precisa ser completamente exato para garantir uma solução aceitável do SCF. Na verdade, o que basta para atingirmos uma solução aceitável é apenas aniquilar (tornar zero ou com valor muito pequeno) todos os elementos da matriz de **F** que conectam os elementos do espaço dos orbitais virtuais ao espaço dos orbitais ocupados. Stewart e colaboradores atestam que esse passo não precisa ser exato justamente porque o próprio algoritmo SCF é robusto. Esse estratégia computacional, chamada de pseudodiagonalização, é mais eficiente que a diagonalização completa, já que não gera um conjunto de autovalores, só novos autovetores. As etapas de pseudodiagonalização são apresentadas à seguir:

- Calcular a matriz $\mathbf{F}_{o-v}$(bloco ocupado-virtual da matriz de Fock), através de$\mathbf{F}_{o-v} = \mathbf{C}_o^{\mathrm{T}}\mathbf{F}\mathbf{C}_v$, onde **F**, $\mathbf{C}_o$e $\mathbf{C}_v$ são respectivamente a matriz de Fock e os autovetores ocupados e virtuais da iteração anterior.
- Calcular os ângulos de rotação α e β, entre o autovetor ocupado *i* e o autovetor virtual *a*, segundo as seguintes expressões:

  $$\alpha = (\epsilon_i - \epsilon_a)^{-1}\mathbf{F}_{o-v_{ia}} \mathrm{e} \beta = \sqrt{(1-\alpha^2)}$$

  onde,ϵ_i eϵ_a são os respectivos autovalores dos orbitais moleculares *i* e *a*, e $\mathbf{F}_{ia}$ é o elemento (i,a) da matriz $\mathbf{F}_{o-v}$que será zerado ou tornado muito pequeno.
- Realizar rotações (denominadas de transformações de Jacobi) entre os autovetores ocupados e virtuais *i* e *a*, de forma a gerar novos autovetores para o cálculo das iteração seguinte usando às seguintes expressões:

  $$\tilde{C}_i = \alpha C_i - \beta C_a \quad \mathrm{e} \quad \tilde{C}_a = \alpha C_a + \beta C_i$$

Usualmente, a pseudodiagonalização é 50% mais rápida que uma diagonalização completa, e, se levado em consideração a esparsidade de uma matriz, podemos usá-la para alcançar um escalonamento quase-linear no tempo de execução.

2. Metodologia

No algoritmo original(Stewart et al., 1982), a ordem em que os autovetores são rotacionados é importante, pois essas operações afetarão as rotações seguintes, o que força com que as rotações sejam feitas em uma ordem bem determinada, dificultando sua paralelização. Para resolver esse problema, empregamos a proposta de Beierlein e Clark. (Beierlein & Clark, 2005). O pseudocódigo mostrado no esquema 1 reflete o funcionamento do algoritmo.

```
1. Inicia as variáveis tiny = 10^-7, i = 1;
lumo = NumOrbsOcupados+1;
j = lumo,ij = 0;
2. Enquanto i <= NumOrbsOcupados faça:
a. Enquanto j <= NumOrbitaisTotal faça:
i.    ij++;
ii.   x = Fmo(ij);
iii.  Se (módulo(x) <tiny) goto x;
iv.   A = autovalor(i);
v.    B = autovalor(j);
vi.   alpha = (b-a)^-1*x;
vii.  beta  = sqrt(1-alpha^2);
viii. coef.alpha[i,j-lumo] = alpha;
ix.   coef.beta[i,j-lumo]  = beta;
x.    j++;
b. Fim do laço a;
c. i++;
3. Fim do laço 2;
4. Fim;
```

(a)

```
1.Inicia as variáveis i = 1;
lumo = NumOrbsOcupados+1;
j = lumo, ij = 0;
2.Ci0 = Ci
3.Ca0 =
4.Enquanto i <= NumOrbsOcupados faça:
a.Enquanto j <= NumOrbitaisTotal faça:
i.   alpha = coef.alpha[i,j-lumo];
ii.  beta  = coef.beta[i,j-lumo]
iii. Ci = alpha*Ci0 - beta*Ca0;
iv.  Ca = alpha*Ca0 + beta*Ci0;
v.j++;
b.Fim do laço a;
c.i++;
5.Fim do laço 4;
6.Fim;
```

(b)

Esquema 1. Pseudocódigos da (a) cálculo dos ângulos de rotação α e β e (b) rotação dos autovetores de F.

2.1. Implementação na GPU

A plataforma CUDA (*Computer Unified Device Architecture*) é uma plataforma de programação paralela, que nos dá acesso ao poder de processamento das placas gráficas para propósito geral. Por natureza, a GPU é um coprocessador extremamente paralelo, com muitos núcleos, então, podemos utilizar esse poder para desenvolver aplicações híbridas, que utilizam a GPU em partes computacionalmente custosas de programas.

Nossa implementação consistiu de dois *kernels* diferentes: um para computar os coeficientes de rotação α e β dos autovetores, e outro para efetivamente realizar as rotações. Da forma como o algoritmo foi proposto, primeiramente podemos calcular os ângulos de rotação e rotacionar o vetor no mesmo *kernel*, porém, alguns elementos do bloco ocupado-virtual são muito pequenos, e não é necessário que haja uma série de rotações para eliminá-lo. Então, se computarmos primeiro os coeficientes, eliminamos

divergências entre as *threads* da GPU no passo da rotação, visto que só vamos calcular as rotações que realmente devem ser feitas.

3. Resultados

Os cálculos foram realizados em um computador com a essa especificação: dual Intel Xeon E52620 (seis cores cada) com 64GB de RAM, Tesla K20c com 5GB de memória dedicada. O sistema operacional foi Ubuntu 12.10. Os compiladores para Fortran e para C++ foram respectivamente o Intel Ifort 13.1.3 e o NVCC do CUDA toolkit v.5.5.

O número de rotações na pseudodiagonalização de **F** que cada iteração do SCF faz pode variar, já que nem todas as rotações devem ser realizadas. Então, para comparar os resultados de *speedup*, nós temporizamos a etapa de rotação em cada iteração para ambos os procedimentos na CPU e na GPU.

Para testarmos a performance dos nossos algoritmos paralelos construímos sistemas moleculares contendo de 2000 até 11600 orbitais moleculares e realizamos os cálculos de química quântica. Para esses sistemas, conseguimos *speedups* medianos (entre 3 e 6 vezes mais rápido). Uma provável razão pela qual obtivemos esses resultados pode ser devido ao alto número de acessos a memória global, já que esse algoritmo é de difícil paralelização mesmo com adaptações.

Nesse trabalho aceleramos a etapa da rotação de autovetores para a pseudodiagonalização da matriz de Fock no SCF, quando usamos métodos semiempíricos de química quântica. Dessa forma, conseguimos suavizar o tempo de execução da rotação na etapa de pseudodiagonalização da matriz **F**.

4. Referências bibliográficas

Beierlein, F., & Clark, T. (2005). Computer Simulations of Enzyme Reaction Mechanisms: Simulation of Protein Spectra. In S. Wagner, W. Hanke, A. Bode, & F. Durst (Eds.), *High Performance Computing in Science and Engineering, Munich 2004* (pp. 245–259). Springer Berlin Heidelberg.

Hwu, W. W. (2011). *GPU Computing Gems Emerald Edition* (1 edition.). Burlington, MA: Morgan Kaufmann.

Maia, J. D. C., Urquiza Carvalho, G. A., Mangueira, C. P., Santana, S. R., Cabral, L. A. F., & Rocha, G. B. (2012). GPU Linear Algebra Libraries and GPGPU Programming for Accelerating MOPAC Semiempirical Quantum Chemistry Calculations. *Journal of Chemical Theory and Computation*, *8*(9), 3072–3081.

Rocha, G. B., & Simas, A. M. (2007). Métodos Semi-empíricos de Estrutura Eletrônica em Química Quântica. In N. H. Morgon & K. Coutinho (Eds.), *Métodos de Química Teórica e Modelagem Molecular* (1a ed., pp. 29 – 71). São Paulo: Livraria da Física.

Stewart, J. J. P., Császár, P., & Pulay, P. (1982). Fast semiempirical calculations. *Journal of Computational Chemistry*, *3*(2), 227–228.