UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Thiago dos Santos Pinto

Implementação do algoritmo de Newman para identificação de comunidade em placas gráficas.

Botucatu
2011

Thiago dos Santos Pinto

Implementação do algoritmo de Newman para identificação de comunidade em placas gráficas.

Monografia apresentada ao Instituto de Biociências da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de Botucatu, para obtenção do título de Bacharel em Física Médica.

Orientador: Prof. Dr. Ney Lemke

Botucatu
2011

FICHA CATALOGRÁFICA ELABORADA PELA SEÇÃO TÉC. AQUIS. TRATAMENTO DA INFORM.
DIVISÃO DE BIBLIOTECA E DOCUMENTAÇÃO - CAMPUS DE BOTUCATU - UNESP
BIBLIOTECÁRIA RESPONSÁVEL: *SULAMITA SELMA CLEMENTE COLNAGO*

Santos-Pinto, Thiago.
Implementação do algoritmo de Newman para identificação de comunidades em placas gráficas / Thiago dos Santos Pinto. - Botucatu, 2011.

Trabalho de conclusão de curso (bacharelado - Física Médica) - Instituto de Biociências de Botucatu, Universidade Estadual Paulista, 2011.
Orientador: Ney Lemke
Capes: 31301037

1. Bioinformática. 2. Aprendizagem de máquina.

Palavras-chave: Aprendizagem de máquina; Bioinformática; Identificação de comunidades.

Agradecimentos

Agradeço primeiramente ao meu pais e avós. Por terem me carregado, mas não excessivamente a ponto de eu não aprender a andar. Por terem deixado eu cair, mas apenas tombos que eles soubessem que eu seria capaz de levantar. Por terem deixado eu trilhar meu caminho, mas não sem aquele olhar de preocupação, que traduz todo amor que tive em minha criação.

Estendo estes agradecimento à toda a minha família e amigos. Porque eu sou um pouco de cada um de vocês. Em especial ao meu irmão Rafael, por ter sido uma referência de caráter, capacidade e dedicação; e à minha namorada Camila, pela motivação e o suporte dado a mim.

Agradeço aos meus colegas de república Renato, Gustavo e Éric. Porque não foi fácil chegar até onde chegamos e isso só foi possível porque tínhamos o apoio e a motivação um dos outros.

Agradeço ao meu orientador Ney Lemke pela empenho dedicado à minha formação acadêmica. Seja na atenção dedicada ao ensino da Física Médica, seja no apoio incondicional para qualquer uma de nossas ideia malucas de projetos extracurriculares, quanto no esforço de ensinar-me outros vários assuntos de seu domínio que eu apresentasse algum interesse.

Agradeço aos meus colegas do LBBC por criaram um ambiente de trabalho muito agradável e por toda atenção e auxílio dado a mim .

E, por fim, agradeço a todos os professores que realmente se preocuparam e se dedicaram à nossa formação.

A realização deste projeto só foi possível devido ao apoio do CNPq.

“Quando tínhamos todas as respostas, mudaram as perguntas.”
(frase colhida de um muro de Quito por Eduardo Galeano)

Resumo

Comunidades estão presentes nos mais diversos sistemas sejam eles físicos, químicos ou biológicos, e a identificação destas é fundamental para a compreensão do comportamento desses sistemas. Recentemente, a quantidade de dados relacionado a redes complexas têm crescido exponencialmente, demandando mais poder computacional. As placas de processamento de vídeo (GPUs) são uma alternativa com bom custo-benefício adequada para este propósito. Nós investigamos a conveniência dessa solução para a ciência de redes propondo uma implementação do algoritmo de Newman baseada em GPU para a detecção de comunidades. Mostramos que o tempo de processamento de multiplicação de matrizes em GPU cresce menos rapidamente que o da CPU em relação ao tamanho da matriz. Desta forma, foi demostrado que o processamento em GPU é uma alternativa viável para simulações de identificação comunidades que exijam alta capacidade computacional. Nossa implementação foi testada em uma rede biológica integrada para a bactéria *Escherichia coli*.

Abstract

Communities are present on physical, chemical and biological systems and their identification is fundamental for the comprehension of the behavior of these systems. Recently, available data related to complex networks have grown exponentially, demanding more computational power. The Graphical Processing Unit (GPU) is a cost effective alternative suitable for this purpose. We investigate the convenience of this solution for network science by proposing a GPU based implementation of Newman community detection algorithm. We showed that the processing time of matrix multiplications of GPUs grow slower than CPUs in relation to the matrix size. It was proven, thus, that GPU processing power is a viable solution for community identification simulation that demand high computational power. Our implementation was tested on an integrated biological network for the bacterium *Escherichia coli*.

Índice

1. Introdução

A divisão de sistemas complexos em comunidades pode ser determinante na compreensão de um fenômeno, sendo ele social, físico, químico ou biológico. Na tragédia acontecida no Japão, por exemplo, houve um colapso do sistema elétrico do nordeste devido à destruição causada pelo Tsunami. A parte sudoeste, inafetada pelo desastre, porém, não pôde auxiliar na recuperação da energia na parte afetada devido à divisão do sistema elétrico do país em duas comunidades (50Hz e 60Hz) incompatíveis uma com a outra, como mostra a Figura 1.

Figura 1: Divisão em comunidade do sistema elétrico Japonês. Em azul está a comunidade que opera a 60Hz e em vermelho a que opera em 50Hz [1].

A estrutura de comunidades é obtida por meio de grafos, que são um conjunto de vértices, representando o objeto de estudo, conectados por arestas quando existe interação entre esses vértices. Grafos podem ser direcionados, quando a interação entre os vértices conectados não é mútua, ou não direcionados, quando a interação entre os vértices é mútua, exemplificado na Figura 2 [2].

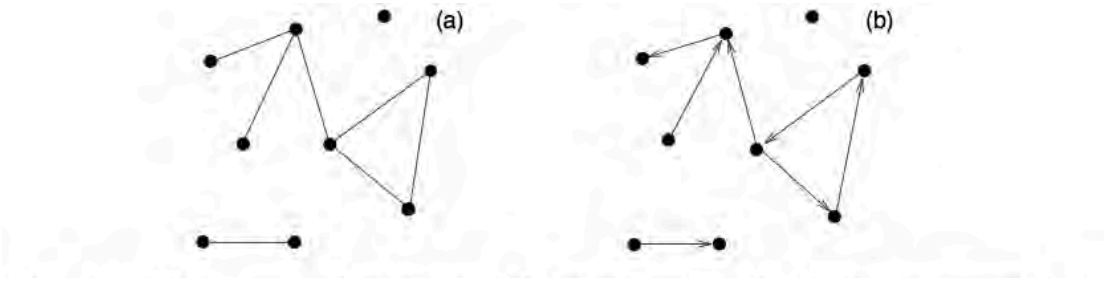

Figura 2: Exemplo de redes. (a) Grafo não direcionado. (b) Grafo direcionado [2].

Comunidades são definidas como grupos de nodos que estão mais densamente interligados entre si do que com outros elementos da rede [3] e tal característica tem sido muito aplicada na biologia de sistemas. Essa área da biologia estuda as interações de redes de genes, proteínas e metabólicas de forma integrada, com objetivo de obter uma compreensão mais robusta do funcionamento dos seres vivos [4]. Nesse panorama a identificação de comunidades tem desempenhado um papel fundamental por representarem unidades funcionais [5]: no grafo de interações entre proteínas, por exemplo, comunidades agrupam proteínas que têm funções semelhantes ou que participam de processos semelhantes, como mostra a Figura 3 [6]. Além disso, estudos indicam que as propriedades de uma comunidade são diferentes das propriedades da rede como um todo, de forma que a análise da rede ignorando a estrutura de comunidades pode resultar na não identificação de fenômenos importantes [5].

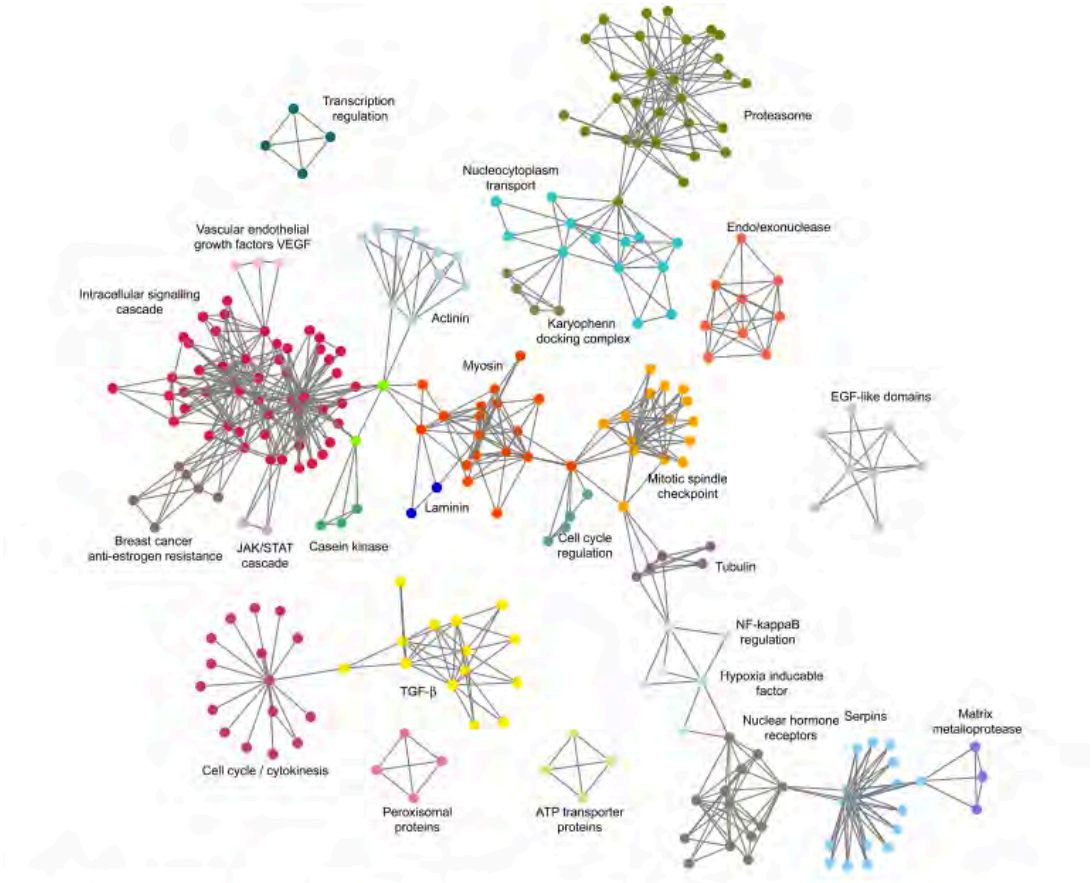

Figura 3: Exemplo de comunidades na interação proteína-proteína [6].

Com o passar do tempo, a quantidade de dados empíricos em bancos de dados sobre o funcionamento de sistemas biológicos têm crescido vertiginosamente, permitindo, assim, a realização de simulações computacionais cada vez mais robustas. Para realizar essas simulações em tempo hábil, porém, a capacidade computacional exigida é cada vez maior.

A solução encontrada para contornar a questão foi utilizar GPUs (*Graphics processing unit*). Devido ao alto investimento na computação gráfica pela indústria de jogos, que necessita de processamento paralelo massivo para a renderização de vídeo, as placas de vídeos se tornaram uma opção muito atrativa para computação de alto desempenho no meio científico. Elas propiciam alto poder computacional com um custo muito menor quando comparado aos agregados computacionais. Tendo isso em vista, a Nvidia, fabricante de GPUs, criou uma arquitetura paralela de propósito geral chamada de CUDA (*Compute Unified Device Architecture*). Essa arquitetura estende a linguagem C com um

conjunto de instruções que permite a programação paralela de suas placas de vídeo [7]. A evolução do número de operações de ponto-flutuante por tempo das GPUs comparado com as CPUs (*Central Processing Unit*) é mostrada na Figura 4.

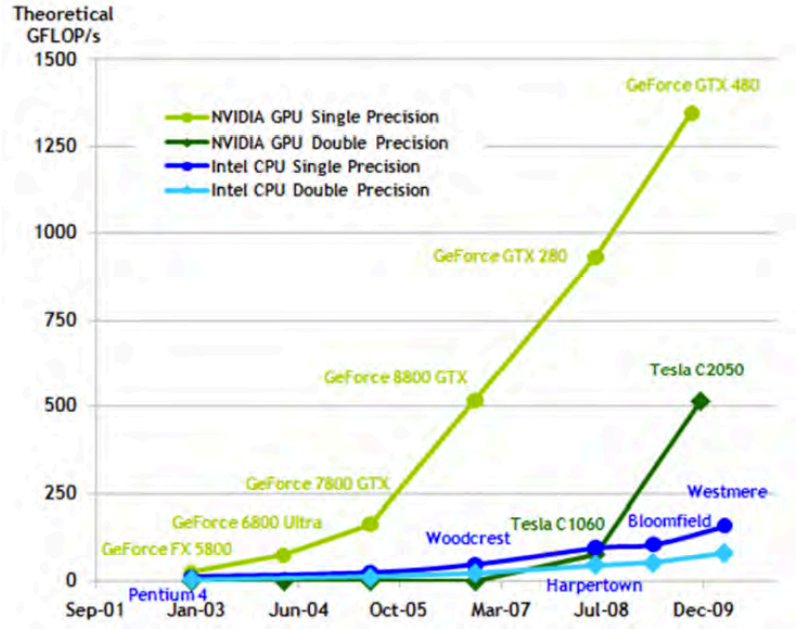

Figura 4: Desempenho das GPUs comparado à CPU no cálculo usando ponto-flutuante [8].

Newman [5] desenvolveu um algoritmo para calcular a divisão em comunidades de um grafo baseada em manipulações matriciais. Essa característica torna o algoritmo paralelizável, sendo ideal para ser utilizado no processamento com placas de vídeo.

Neste trabalho, foi implementado o algoritmo de Newman em placas de vídeo utilizando o CUDA por meio do software Mathematica, que é ideal para o estudo de redes devido à facilidade que ele propicia na manipulação e visualização de grafos.

2. Objetivo

Implementar o algoritmo de processamento paralelo em CUDA para identificação de comunidades de Newman em placas de vídeo utilizando o Mathematica.

2.1 Objetivo parcial

Verificar se a multiplicação de matrizes em GPU é mais eficiente que em CPU no Mathematica.

3. Metodologia

3.1 Hardware e Softwares

Os softwares utilizados para as simulações foram:

- Linux Ubuntu versão 10.10;
- Mathematica para Linux 64 bit 8.0.0.0;
- driver gráfico da Nvidia versão 260.19.26;
- toolkit para CUDA versão 3.2.

A máquina utilizada era equipada com uma placa de vídeo Tesla T10, processador Intel Xeon E5520 de 2.27GHz e 23GB de memória RAM.

3.2 Algoritmo de Newman[5]

Define-se matriz de adjacência **A** como:

$$A_{ij} = \begin{cases} 1 & se\ i\ e\ j\ conectados \\ 0 & caso\ contrário \end{cases} \quad (1)$$

O número de nodos k_i conectados ao vértice i, também chamada do grau do vértice, é dado por:

$$k_i = \sum_j A_{ij} \quad (2)$$

P_i é o número de arestas entre i, j, e é dado por:

$$P_{ij} = \frac{k_i k_j}{2m} \quad (3)$$

sendo m número de arestas da rede.

Para realizar a divisão em comunidades calcula-se **B**, que é uma matriz real e simétrica tendo elementos:

$$B_{ij} = A_{ij} - P_{ij} \quad (4)$$

Então, obtêm-se o autovetor $\mathbf{u_1}$ correspondente ao maior autovalor da matriz B. Esta etapa é implementada de forma paralela na placa de vídeo e, também, é utilizado a função do Mathematica que faz esse cálculo na CPU como comparação.

Com base no autovetor, cria-se um vetor de índices s_i definido por:

$$s_i = \begin{cases} +1 \; se \; u_i^{(1)} \geq 0 \\ -1 \; se \; u_i^{(1)} \leq 0 \end{cases} \tag{5}$$

sendo que elementos do grafo cujo índice s_i é positivo pertencem a uma comunidade e elementos cujo índice s_i é negativo pertencem a outra.

3.3 Modularidade

A modularidade Q é calculada por:

$$Q = \frac{1}{4m} \boldsymbol{s}^T \boldsymbol{B} \, \boldsymbol{s} \tag{6}$$

sendo **s** definido na Equação 5.

Para divisões sucessivas em mais que 2 comunidades é necessário calcular o ΔQ adicional para as divisões subsequentes. Nesse caso o conjunto de vértices a ser dividido será denotado por G e n_G será o número de vértices nessa comunidade:

$$\Delta Q = \mathrm{Tr}(\mathbf{S}^T \boldsymbol{B}^{(G)} \boldsymbol{S}) \tag{7}$$

em que **S** é uma matriz índice n_G x c denotando a subdivisão em c subcomunidades tal que:

$$S_{ij} = \begin{cases} 1 \; se \; o \; vértice \; 1 \; pertence \; \text{à} \; subcomunidade \; j \\ 0 \; caso \; contrário \end{cases} \tag{8}$$

e $\mathbf{B}^{(G)}$ é dado por:

$$B_{ij}^{(G)} = B_{ij} - \delta_{ij} \sum_{l \in G} B_{il} \tag{9}$$

3.4 Cálculo do maior autovetor

Para se obter o maior autovetor de uma matriz simétrica **B**, gera-se um vetor aleatório **u** que é multiplicado à matriz **B**, sendo o resultado normalizado. Esse procedimento é iterado até que o erro para todos os elementos da matriz seja menor que um patamar pré-definido.

$$\boldsymbol{u}_{n+1} = \frac{\boldsymbol{B}\,\boldsymbol{u}_n}{\|\boldsymbol{B}\,\boldsymbol{u}_n\|} \tag{10}$$

O erro é calculado por:

$$e_i = \left|u_i^n - u_i^{n+1}\right| \tag{11}$$

Neste trabalho, o critério de parada da iteração é quando o erro for menor que 10^{-4} para todos os elementos do vetor.

3.5 Método alternativo para o cálculo do maior autovetor

Neste método, a placa gráfica faz a multiplicação $\mathbf{B}^2$, na Equação 12, e a CPU faz a multiplicação matriz vetor na Equação 13. O erro e o critério de parada são os mesmo que o do método anterior.

$$\boldsymbol{B}_{n+1} = \frac{\boldsymbol{B}_n^2}{B_i^{max}} \tag{12}$$

$$\boldsymbol{u} = \frac{\boldsymbol{B}_n\,\boldsymbol{u}}{\|\boldsymbol{B}\,\boldsymbol{u}\|} \tag{13}$$

3.6 Obtenção de comunidades

Para gerar as comunidades, o algoritmo subdivide o grafo inicial em duas comunidades e após isto, cada comunidade é sucessivamente subdividida em duas outras, como exemplificado na Figura 5.

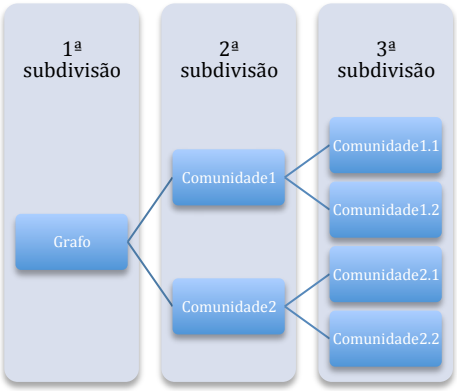

Figura 5: Exemplo da divisão hierárquica do algoritmo utilizado para a identificação de comunidades.

O critério utilizado para verificar a possibilidade de subdividir uma comunidade foi a modularidade da mesma, descrito no item 3.7.

3.7 CUDA

Um programa CUDA consiste em código fonte unificado compreendendo código do host e do *device*. O código do host é escrito em ANSI (*American National Standards Institute*) C simples e o código do *device* é escrito usando ANSI C estendido com palavras-chave para rotular as funções com dados paralelos, chamados *kernels*, e suas estruturas de dados associadas [7].

O *kernel*, quando chamado, executa as instruções contidas dentro da função paralelamente em N *threads* diferentes. Os *threads* são agrupados em blocos que, por sua vez, são agrupados em um *grid*, como mostra a Figura 6 [8].

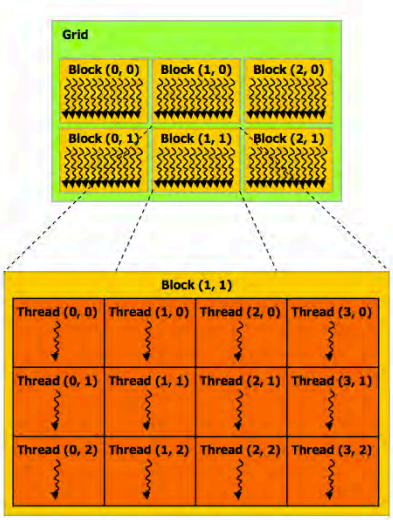

Figura 6: Ilustração da estrutura de processamento em GPUs [8].

Cada bloco é processado por um core do multiprocessador, portanto, *threads* de um mesmo bloco podem interagir entre si por meio da memória compartilhada. Os blocos criados são colocados em uma fila de execução, sendo executados independentemente uns do outros e de acordo com a capacidade de processamento da GPU, como ilustra a Figura 7 [8].

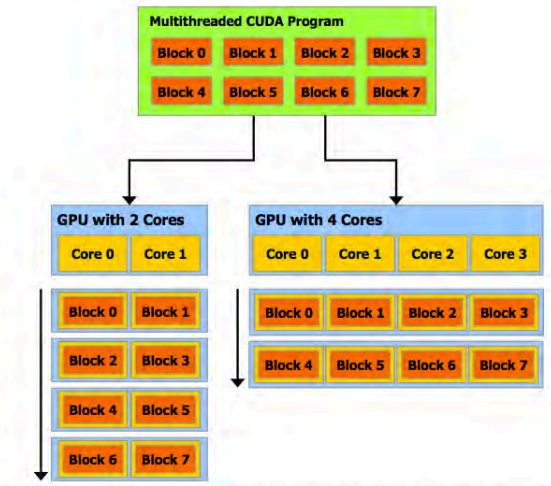

Figura 7: Ilustração da execução dos blocos de acordo com o número de núcleos disponíveis [8].

3.8 Mathematica

O Mathematica, *software* produzido pela Wolfram, é um sistema computacional que inclui uma linguagem de programação funcional. A partir da

versão 8, foram criadas funções que utilizam a linguagem CUDA para ter acesso ao processamento da GPU. A vantagem de se utilizar o CUDA por meio do Mathematica é não ter que se preocupar com criação do *kernel* e com o gerenciamento de memória, fatores que consomem tempo considerável da elaboração do programa em CUDA. A função do Mathematica, que realiza multiplicação matricial em GPU, utilizada nos testes é o CUDADot.

3.9 Teste de velocidades com multiplicação de matrizes

Foram feitos testes comparativos entre o tempo de processamento de multiplicação matrizes quadradas de tamanhos 1000 até 10000 passo 500. Para cada passo a matriz aleatória gerada foi multiplicada com ela mesma usando a CPU e a GPU.

Além disso, também foi feito o teste para verificar o tempo de processamento das matrizes acima com um vetor do mesmo tamanho.

3.10 Regressão não-linear dos dados

Foi realizado a regressão não-linear para dados obtidos com resultado dos testes de velocidade realizados no item 3.4 segundo a Equação 14 .

$$t = by^{\gamma} \tag{14}$$

sendo t o tempo de processamento e y o tamanho da matriz.

4. Resultados e Discussão

4.1 Teste de velocidade na multiplicação de matrizes

A comparação entre o tempo de processamento da CPU e GPU para multiplicação de matrizes é dado no Figura 8. As funções da regressão não-linear para os dados da GPU e CPU são dados na equações 15 e 16, respectivamente.

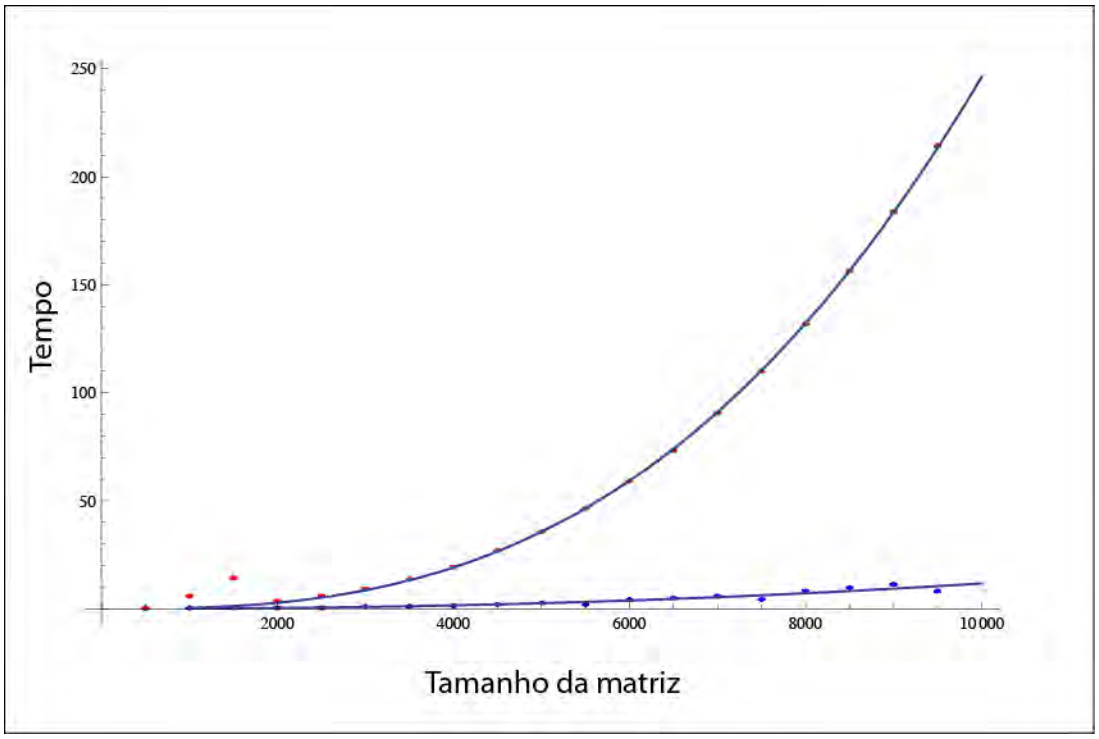

Figura 8: Gráfico da comparação do tempo de multiplicação de duas matrizes em GPU (Azul), CPU (Vermelho). Foram repetidas 10 vezes as simulações. As barras de erro são da mesma ordem que os pontos da figura.

$$t_{GPU} = 1{,}97\ x\ 10^{-8}\ y^{2{,}20} \quad (15)$$

$$t_{CPU} = 1{,}73\ x\ 10^{-9}\ y^{2{,}79} \quad (16)$$

Foi possível realizar cálculos utilizando matrizes com tamanho de até 16000 unidades. Matrizes com valores acima deste, excederam a capacidade de memória disponível na GPU. No gráfico foram apresentados os resultados de tempo apenas para matrizes de tamanho até 10000. Isso porque não havia

disponível a capacidade computacional para repetir, em tempo hábil, a simulação no mínimo 10 vezes.

Como se pode observar pela Figura 8, a GPU provou ser mais eficiente que a CPU para multiplicação de matrizes. E, segundo o valor do γ das equações 16 e 17, quanto maior a matriz processada, mais eficiente é a GPU em relação à CPU.

A comparação entre o tempo de processamento da CPU e GPU para multiplicação matriz-vetor é dado no Figura 9. As funções de regressão não-linear para os dados da GPU e CPU são dados na equações 17 e 18, respectivamente.

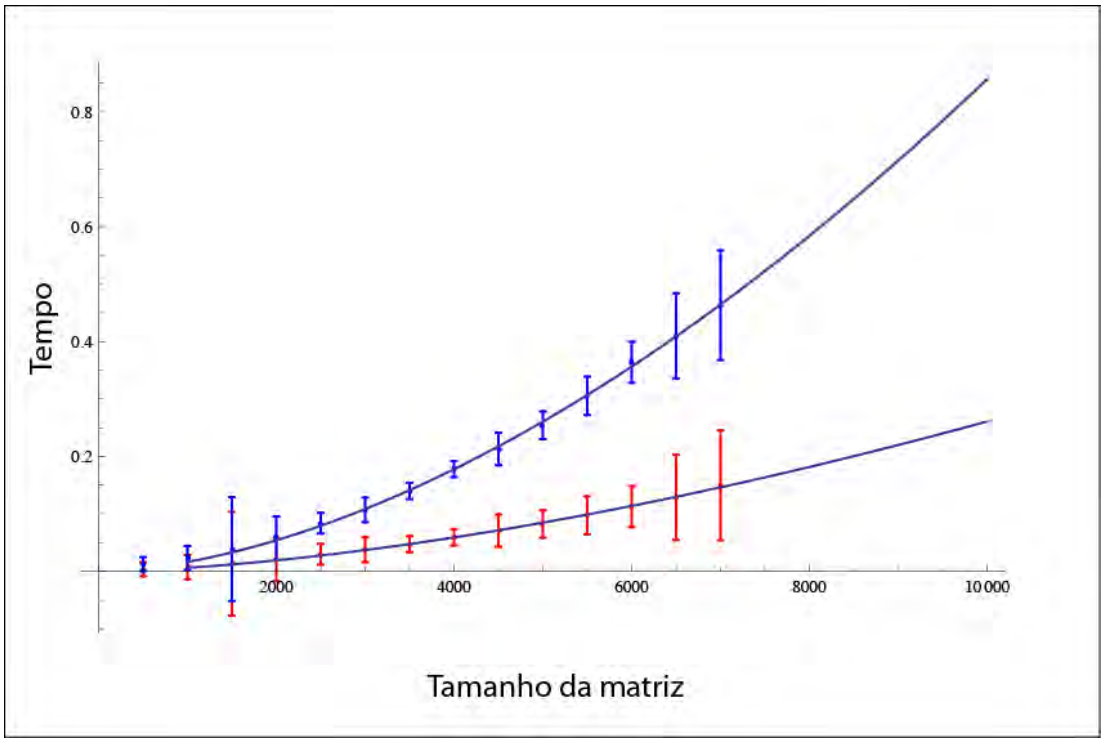

Figura 9: Gráfico da comparação do tempo de multiplicação entre uma matriz e um vetor na GPU (Azul) e na CPU (Vermelho). Foram repetidas 10 vezes as simulações.

$$t_{GPUMV} = 1{,}17\ x\ 10^{-7}\ y^{1{,}72} \quad (17)$$

$$t_{CPUMV} = 8{,}49\ x\ 10^{-8}\ y^{1{,}62} \quad (18)$$

Para a multiplicação matriz-vetor, não foi possível processar matrizes superiores a 7500 unidades devido à falta de memória na GPU.

A Figura 9 indica que a GPU não apresenta maior eficiência que a CPU para multiplicação matriz-vetor. Os valores de γ nas equações 17 e 18 indicam, ainda, que para esse tipo de operação a GPU será menos eficaz que a CPU, independente do tamanho do conjunto matriz-vetor. A razão da ineficiência da função com operações desta natureza pode estar ligado ao fato de ela ter sido projetada especificamente para operações matriz-matriz. O que justifica, também, a incapacidade de alocar corretamente a memória na GPU.

4.2 Algoritmo do Newman

Tendo em vista os dados referentes à multiplicação matriz-vetor, a metodologia para obter o maior autovalor foi reformulada para utilizar multiplicação matriz-matriz, conforme descrito na metodologia.

Além das vantagens discutidas anteriormente, outro ponto positivo deste novo método é que ele converge para um resultado com no máximo 5 iterações, enquanto o método anterior leva até 500 iterações para convergir.

A figura 8 mostra rede compilada por V. Krebs (não publicado) que representa pares de livros políticos [9] mais frequentemente comprada pelos mesmos consumidores.

Figura 8: Grafo da rede de livros políticos mais frequentemente comprado pelos mesmo consumidores [9]. Comunidades geradas pelo algoritmo implementado.

A divisão em comunidades, da rede da figura 8, foi realizada utilizando o algoritmo proposto neste trabalho. Foram identificadas, assim como no trabalho de Newman [5], duas grandes comunidades, representando livros com ideologia de liberal(laranja) e conservadora(vermelho). O tempo de processamento para a identificação das comunidades desta rede foi desprezível.

A figura 9 mostra um grafo contendo as interação metabólica, regulatória e proteína-proteína da bactéria *Escherichia coli*, dividido em comunidades pelo algoritmo implementado neste trabalho.

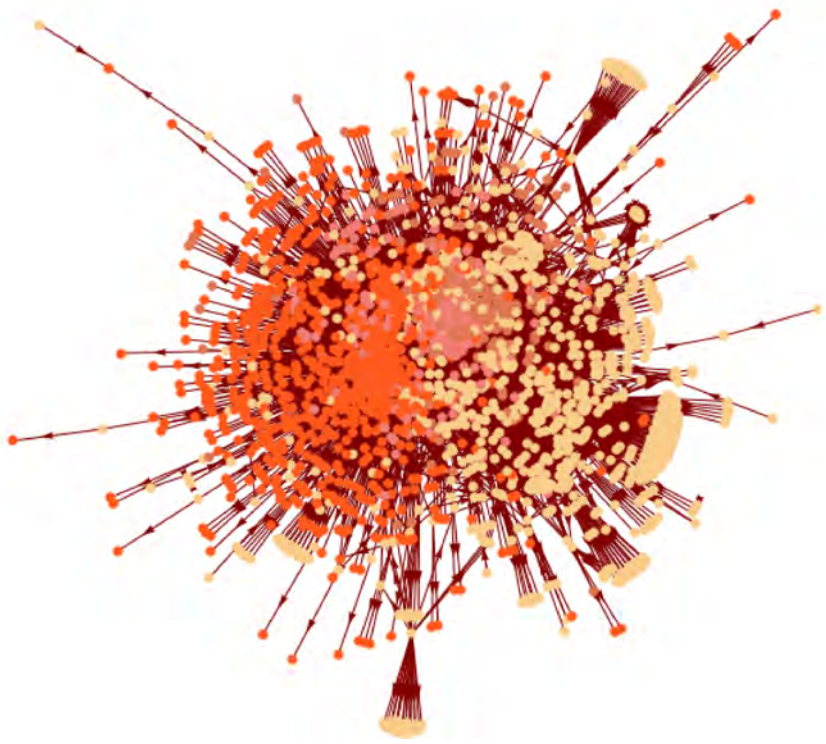

Figura 9: Grafo das interações metabólica, regulatória e proteína-proteína da *Escherichia coli*, dividido em comunidades pelo algoritmo implementado.

Para a rede da figura 9 a modularidade foi ajustada de modo a se obter 4 comunidades. Isso se deve ao fato que a detecção comunidades, além das 4 primeiras obtidas, não são claramente visualizada devido à complexidade da rede.

A tabela 1 mostra a comparação entre os tempos previsto pelas funções de regressão não linear e o tempo obtido ao se realizar a multiplicação da matriz de adjacência da rede da figura 9.

Tabela 1: Comparação entre o tempo previsto pela função de regressão não-linear e o tempo obtido na multiplicação da matriz de adjacência da rede da *Escherichia coli*.

	Regressão não-linear (s)	Medido (s)
GPU	$5{,}48 \times 10^{-1}$	$(2{,}5 \pm 0{,}7) \times 10^{-4}$
CPU	4,66	$(5{,}62 \pm 0{,}01) \times 10^{-3}$

A grande diferença encontrada na tabela 1, é devido à forma como foram coletados os dados para a regressão não linear. Para a coleta destes dados, foram geradas novas matrizes, e com tamanho diferentes, a cada passo da iteração. No caso da matriz de adjacência da rede da E. Coli, era uma única matriz que foi iterada 20 vezes para se obter os tempos com o desvio padrão. Dessa forma, a diferença entre os tempo pode estar relacionado ao tempo de carregamento da memória que, no caso do teste com a matriz de adjacência, foi zero para todos os passos da iteração exceto o primeiro. Além disso, o fato da matriz de adjacência ser composta de muito mais zeros que a matriz aleatória utilizada nos teste para a regressão não-linear pode ser outro fator que influenciou na diferença dos tempos na tabela 1.

5. Conclusão

O desempenho do algoritmo de Newman em GPU não foi mais eficiente que em CPU quando realizado com multiplicação matriz-vetor. Já o novo método proposto, utilizando multiplicação matriz-matriz, demonstrou que a GPU é mais eficaz que a CPU no processamento, principalmente para matrizes maiores.

Para a rede de livros políticos, as comunidade obtidas com o algoritmo desenvolvido neste trabalho foram iguais às comunidade obtidas no trabalho de Newman [5].

Como possível extensão a este trabalho propomos criar um *kernel* CUDA específico para multiplicações matriz-vetor e substituí-lo pela função nativa do Mathematica utilizado na implementação do algoritmo de Newman.

6. Referências bibliográficas

[1] Fischetti M. Japan's two incompatible power grids make disaster recovery harder [Internet. Atualizada em 25 de Março de 2011, acesso em 06 de Junho de 2011]. © 2011 Scientific American, a Division of Nature America, Inc. Disponível em:
http://www.scientificamerican.com/blog/post.cfm?id=japans-two-incompatible-power-grids-2011-03-25&WT.mc_id=SA_DD_20110325.

[2] Dorogovtsev SN, Mendes JFF. Evolution of networks : from biological nets to the Internet and WWW. Oxford ; New York: Oxford University Press; 2003.

[3] Barabasi AL, Oltvai ZN. Network biology: understanding the cell's functional organization. Nat Rev Genet. 2004 Feb;5(2):101-13.

[4] Kitano H. Computational systems biology. Nature. 2002 Nov 14;420(6912):206-10.

[5] Newman ME. Finding community structure in networks using the eigenvectors of matrices. Phys Rev E Stat Nonlin Soft Matter Phys. 2006 Sep;74(3 Pt 2):036104.

[6] Jonsson PF, Cavanna T, Zicha D, Bates PA. Cluster analysis of networks generated through homology: automatic identification of important protein communities involved in cancer metastasis. BMC Bioinformatics. 2006;7:2.

[7] David K, Hwu W W. Programando para processadores paralelos: uma abordagem prática à programação de GPU. Tradução de "*Programming massively parallel processors"* por Daniel Vieira. Rio de Janeiro: Elsevier, 2011.

[8] NVIDIA Corporation. NVIDIA CUDA Programming Guide, 22/10/2010. Version 3.2. Disponível em:
http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Programming_Guide.pdf.

[9] Newman ME. Network data [Internet]. Ann Arbor: University of Michigan. [Atualizado em 14 de Junho de 2011, acesso em 15 de Junho de 2011]. Disponível em: http://www-personal.umich.edu/~mejn/netdata/