ciência da computação

V EPAC - Encontro Paranaense de Computação ISSN:1981-8653
18 a 20 de setembro de 2013
Unioeste – Cascavel – Paraná

Otimização de Memória no Alinhamento de Genoma em GPU/CUDA

Nilton L. Queiroz Jr [1], Henrique Y. Shishido[2], Ronaldo A. L. Gonçalves[3]
[1,3] Universidade Estadual de Maringá – UEM – Maringá – Paraná
[1]**niltonluizqueirozjunior@hotmail.com;** [3]**ralgonca@din.uem.br**
[2]Universidade Tecnológica Federal do Paraná – UTFPR – Cornélio Procópio – Paraná
[2]**shishido@utfpr.edu.pr**

Resumo

Os computadores convencionais evoluíram para sistemas computacionais que agregam recursos de paralelização, tais como as unidades de processamento gráfico (GPU). Nessa tecnologia, aplicações paralelas podem ser facilmente desenvolvidas usando bibliotecas disponíveis e uma linguagem de programação convencional, entretanto, o desempenho depende da forma como os dados são alocados e acessados na memória da placa GPU. O algoritmo de alinhamento global de Needleman-Wunsch é muito importante na comparação de sequências de DNA, e o presente trabalho propõe a paralelização de tal algoritmo na arquitetura GPU usando a biblioteca CUDA. Duas abordagens diferentes de alocação dos dados na memória da GPU são experimentadas. Os resultados mostram que armazenar os dados matriciais no sentido das colunas provê um speedup de 1.57x sobre o modelo de linhas.

Palavras-chaves — metodologia de paralelização, plataforma GPU/CUDA, alinhamento de genoma;

1. Introdução

As unidades de processamento gráfico, ou GPU (Cook, 2012), são placas gráficas de alto poder de processamento que têm sido usadas na execução paralela de aplicações de propósito geral, por meio de uma plataforma de programação chamada de CUDA (Nickolls, 2008). Usando GPU/CUDA, aplicações paralelas podem ser desenvolvidas para solucionar uma gama de problemas em diferentes áreas científicas, a um excelente custo-benefício.

GPUs são processadores compostos por diversos núcleos, podendo conter milhares de núcleos de processamento, organizados em grupos chamados de *Streaming Multiprocessors* (SMs) (Chatterjee, 2011), os quais são responsáveis por escalonar e executar grupos de 32 *threads* (*warps*). CUDA (NVIDIA GF100, 2010) é uma plataforma que oferece extensão para varias linguagens de programação convencionais para atuar em GPUs. O programador escreve uma aplicação em linguagem convencional e faz chamadas a *kernel*s CUDA, que podem codificar simples funções ou até programas inteiros. A aplicação executa no hospedeiro, mas os *kernel*s executam na placa GPU por meio de um conjunto de *threads* paralelas.

Muitas aplicações têm sido paralelizadas sobre a plataforma GPU/CUDA, por exemplo, no campo da filogenética (Ying, 2011), na cardiologia em tempo real (Asen, 2012), identificação de tumores (Gu, 2010), detecção de terremotos (Meng, 2012), simulação de tsunamis (Kardos, 2013), previsão financeira (Creel, 2012), detecção de humanos (Farhadi, 2011), dinâmica de fluidos (Lin, 2011) e propagação de ondas (Zhou, 2012), entre outros. O alinhamento de sequências se encaixa nesse contexto, sendo o algoritmo de *Needleman-Wunsch* (Needleman, 1970; Martins, 2001) amplamente usado para calcular a similaridade entre sequências de proteínas ou DNA. Pelo fato do tempo de execução ser muito alto para longas sequências, o presente trabalho propõe sua paralelização sobre a plataforma GPU/CUDA, utilizando duas abordagens de alocação de dados.

2. Algoritmo de *Needleman-Wunsch*

O algoritmo de alinhamento global de *Needleman-Wunsch* tenta encontrar segmentos similares entre duas sequências biológicas, pontuando-os com um escore que mede o grau de similaridade entre eles. O algoritmo analisa todos os possíveis segmentos existentes dentro das duas sequências por meio de uma matriz de similaridade. Cada linha da matriz é associada a um caractere da sequência A e cada coluna a um caractere da sequência B, de forma que a quantidade de elementos da matriz equivale ao produto dos tamanhos das sequências comparadas. Tal matriz serve para realizar todas as possibilidades de comparação entre os caracteres das sequencias A e B. A matriz é preenchida na medida em que é percorrida e as entradas recém-preenchidas são utilizadas nos cálculos das próximas entradas e assim sucessivamente.

O elemento (i,j) da matriz é preenchido com o escore ótimo do alinhamento entre os i primeiros elementos de A e os j primeiros de B. As primeiras linha e coluna da matriz são inicializadas com valores inteiros decrescentes, iniciando em 0 e decrescendo linearmente dependendo do valor da penalidade de gap. Os demais elementos da matriz são calculados. O cálculo de cada elemento envolve mais 3 elementos vizinhos imediatos, que juntos formam uma sub-matriz 2x2. Os vizinhos do elemento (i,j) são: o vizinho "acima" (i-1,j), o vizinho à "esquerda" (i,j-1) e o vizinho no canto superior esquerdo (i-1,j-1) da sub-matriz mencionada, como exemplificado na Figura 1. O cálculo do valor de substituição dependente da comparação entre os caracteres A_i e B_j e a penalidade de gap, que penaliza os espaços entre os caracteres. Sendo M a matriz de similaridade, GP a penalidade para GAP e ss o escore de substituição, o preenchimento da matriz pode ser dado pela Equação (1) .

$$M[i,j] = \max \begin{Bmatrix} M[i,j-1] + GP \\ M[i-1,j-1] + ss \\ M[i-1,j] + GP \end{Bmatrix} \quad (1)$$

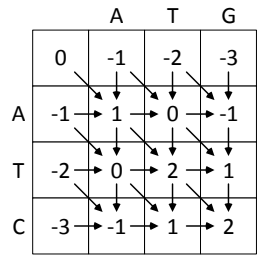

Figura 1: Dependência de dados no algoritmo de *Needleman-Wunsch*

3. Paralelização e Abordagens para a Alocação dos Dados na Memória

O modelo de paralelização aqui desenvolvido prevê o preenchimento da matriz em partes (fatias), pois a matriz não cabe toda na memória da GPU. Cada fatia é tratada por um *kernel* contendo um bloco de *threads* em paralelo. Pela natureza da aplicação, a execução da próxima fatia depende do término da fatia anterior, então, existe uma barreira de sincronização no final de cada fatia. Cada *thread* atua sobre uma linha diferente, caminhando por ela no cálculo do próximo elemento, até o final da linha. Para reduzir dependências entre *threads*, os caminhamentos das *threads* nas linhas foram deslocados de uma posição, criando um fluxo de 1024 *threads* em diagonal na matriz. Essa metodologia de paralelização foi combinada com duas abordagens de alocação da matriz de similaridades na memória. A matriz é

armazenada em um vetor, porém, na primeira abordagem, esse armazenamento é feito de linha por linha, e na segunda, o armazenamento é feito de coluna por coluna, conforme mostra a Figura 2.

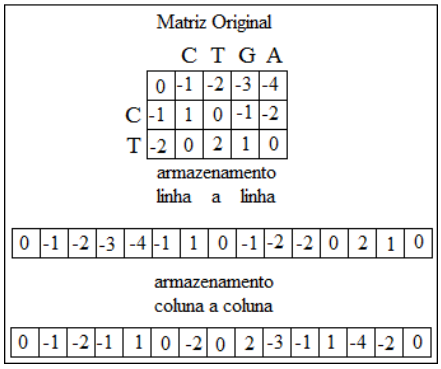

Figura 2: métodos de armazenamento da matriz de similaridade

4.Experimentos e Avaliação

Os experimentos foram executados em um computador com dois processadores Intel(R) Xeon(R) CPU E5-2630 com 32 GB de memória RAM, uma placa GPU GeForce GTX 680 e o sistema operacional Linux Ubuntu 11.10. A referida GPU possui 1536 núcleos de processamento, organizados em 4 GPCs (cluster de processamento gráfico), constituídos de 2 SMXs (SMs de última geração) cada. Os experimentos foram executados com matrizes pequenas, médias e grandes, variando a dimensão *n* do tamanho *nxn* e fixando o numero de *threads* em 1024 (tamanho máximo do bloco na GPU utilizada). Os resultados refletem médias de vários experimentos e alguns são exibidos nas Figuras 3, 4, 5 e 6. A versão sequencial foi experimentada no próprio hospedeiro.

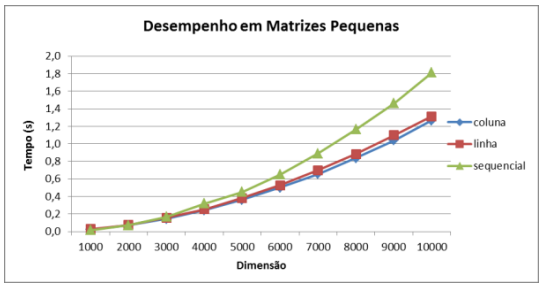

Figura 3: Tempo de execução para matrizes pequenas

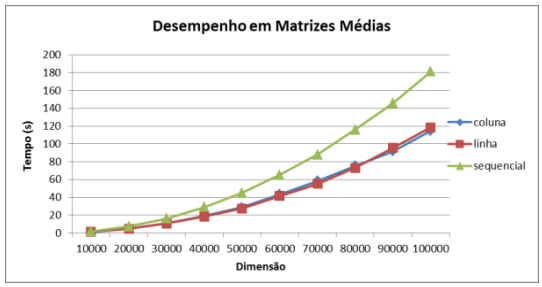

Figura 4: Tempo de execução para matrizes medias

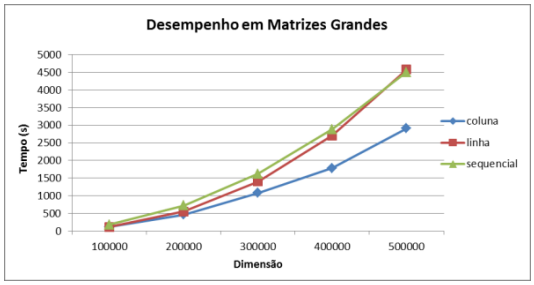

Figura 5: Tempo de execução para matrizes grandes

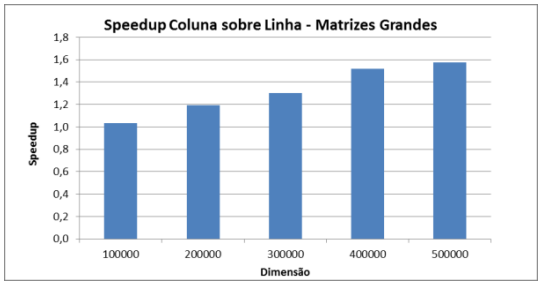

Figura 6: *Speedup* colunas sobre linhas

Pode-se observar que para matrizes de similaridades pequenas e médias, que não estressam muito a memória, os resultados da paralelização em GPU/CUDA, independente da abordagem de alocação dos dados na memória, são bem melhores que aqueles obtidos com a versão sequencial, atingindo redução no tempo de execução em até 37%. Nesses mesmos experimentos houve pouca variação entre as abordagens de alocação utilizadas, ocorrendo uma diferença na redução do tempo de execução, de uma para a outra, na faixa de 4% a 8%. Para matrizes grandes, a cache da GPU sofre pelo aumento de faltas e substituições, mas a abordagem de alocação por coluna se mostrou muito mais eficiente neste caso, pois otimizou o uso da memória.

A grande vantagem desta abordagem, além do melhor desempenho, reside no fato de não requerer qualquer esforço computacional adicional além daquele de reorganizar os dados da matriz de similaridades em função das colunas e não das linhas, como normalmente é feito. O *speedup* da abordagem por colunas sobre a de linhas alcança 1.57x para matrizes de ordem 500000x50000 elementos. Acredita-se que tal vantagem é alcançada pelo fato de uma mesma coluna ser acessada por várias *threads* em momentos próximos, permitindo que estes dados permaneçam por mais tempo na cachê antes de serem substituídos. Além do mais, da forma como foi implementado o processamento da matriz, em "fatias", uma coluna irá ter no máximo 1024 elementos, pois é o limite de um bloco, mas uma linha poderá ter um número bem maior de elementos, dependendo do tamanho das sequências analisadas. Assim, cabem mais colunas do que linhas da matriz na *cache*.

5. Conclusão e Trabalhos Futuros

Nos experimentos aqui realizados, a paralelização em GPU/CUDA do algoritmo de *Needleman-Wunsch* provê um ganho de desempenho considerável em termos de redução do tempo de execução, comprovando o ótimo custo-benefício para este tipo de aplicação. Além disso, uma simples reorganização dos dados na memória pode prover uma melhoria na ordem de 1.57x sobre a própria paralelização, e com possibilidade de ser melhor com o aumento da estrutura de dados. Como trabalhos futuros, pretende-se ainda experimentar o armazenamento da matriz no sentido das diagonais, de forma a aproveitar o fato de que as *threads* caminham com deslocamento sobre as linhas, com o objetivo de reduzir ainda mais a quantidade de faltas na cache durante o preenchimento da matriz.

Referências

CHATTERJEE, S. et al. Dynamic Task Parallelism with a GPU Work-Stealing Runtime System. 24th Int. Workshop on Languages and Compilers for Parallel Computing, 2011.

COOK, S. CUDA Programming: A Developer's Guide to Parallel Computing with GPUs. Morgan Kaufmann, Nov, 2012.

NEEDLEMAN, S.B.; WUNSCH, C.D. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". Journal of Molecular Biology 48 (3) pp. 443, 1970.

NICKOLLS, J. et al. Scalable Parallel Programming with CUDA. Queue, v.6 n.2, March/April 2008.

NVIDIA's Next Generation CUDATM Compute Architecture: FermiTM, Whitepaper, http://www.nvidia.com, 27/07/2013.