UNIVERSIDADE DE SÃO PAULO

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Relatório do Projeto Final - Programação Concorrente

Prof. Dr. Júlio Cezar Estrella

Junho – 2012

Grupo 9 – Turma A

Teorema do macaco infinito – Implementação em CUDA

I.L. Andrade *L. P. dos Santos* *R. Sartori*

Teorema do macaco infinito – Implementação em CUDA

Igor de Lorenzi Andrade[1], Lucas Paulino dos Santos[2], Rafael Sartori[3]

[1] Instituto de Ciências Matemáticas e de Computação, nº USP: 6878824

igordla@grad.icmc.usp.br

[2] Instituto de Ciências Matemáticas e de Computação, nº USP: 6792951

lucasps@grad.icmc.usp.br

[3] Instituto de Ciências Matemáticas e de Computação, nº USP: 6792861

rsartori@grad.icmc.usp.br

Resumo: Visando o aprofundamento do conteúdo da disciplina de Programação Concorrente, este projeto propõe a implementação de uma aplicação em linguagem C que consiga gerar aleatoriamente as palavras contidas em um arquivo. Tal motivação deriva-se do Teorema do macaco infinito, onde se afirma que um macaco escrevendo palavras aleatoriamente pode produzir qualquer obra escolhida, tal como a de Shakespeare. Para o desenvolvimento do conceito, utilizou-se da plataforma CUDA.

Palavras-Chave: Programação Concorrente; Programação Distribuída; CUDA, Macacos Infinitos, Teorema do macaco infinito, Shakespeare.

1. Introdução

A fim de provar o *Teorema do macaco infinito* (explicado adiante), este projeto propõe a criação de uma aplicação capaz de gerar aleatoriamente todas as palavras contidas em arquivo texto base. Tal objetivo é alcançado através da paralelização de tarefas, onde o conceito chave está em acelerar o processo de geração de palavras aleatórias e conferi-las até a garantia de que todas foram geradas. Dessa maneira, torna-se possível demonstrar a validade do teorema deduzindo sua principal premissa. No entanto, isso é realiza em uma quantidade finita de tempo.

O *Teorema do macaco infinito* consiste na afirmação de que um macaco digitando aleatoriamente em uma máquina de escrever por um intervalo infinito de tempo irá certamente recriar um texto escolhido, como por exemplo, a obra completa de Shakespeare.

Estendendo ainda mais as possibilidades de resolução de problemas complexos, neste projeto, além da linguagem C, também fizemos uso da tecnologia *Compute Unified Device Architecture* (CUDA). (Nos projetos anteriores foram usados os conceitos de *threads* POSIX, OpenMP e MPI.) Esta plataforma permitiu tirar proveito das unidades de processamento gráfico (GPUs), uma vez que oferecem maior poder de processamento comparados a CPU para determinadas tarefas.

Além disso, a utilização do CUDA proporcionou alguns benefícios singulares para a resolução do problema, tal como maior quantidade de unidades servindo na geração de palavras aleatórias e, também, maior rapidez na velocidade de comparação dos dados, pois as latências dos envios e recebimentos são muito mais baixas na GPU comparadas as do Cluster.

2. Descrição do Problema

A partir de um arquivo com diversas palavras de natureza e tamanhos variados, o conceito da implementação se dá pela geração totalmente aleatória de cada caractere constituinte de uma palavra, a qual é comparada com toda a base de palavras carregada e filtrada previamente. Através de uma busca, quando uma palavra da base coincide com uma das geradas aleatoriamente, esta deve ser marcada e não será mais aceita nas próximas buscas de comparação. Dessa forma, torna-se necessário que haja constante comunicação entre as unidades de processamento.

Para computação do tempo, as saídas devem demonstrar as marcações parciais de tempo baseadas no volume de palavras processado, expresso em porcentagem. Por exemplo, 10% (X minutos), 20% (X minutos), 30% (X minutos), 40%, 50%, 60%, 70%, 80%, 90% e 100%.

Quanto à filtragem, devem ser levados em conta o tamanho máximo de cinco letras por palavra (o que implica em dividir as maiores que cinco em partes menores) e a desconsideração de espaços e hifens de palavras compostas.

Portanto, tendo em vista os possíveis problemas de *throughput* (quantidade de dados processados em um determinado espaço de tempo) como atomicidade das comparações e atualizações de contadores, a ideia principal é minimiza-los alcançando maior desempenho e menor tempo total gasto com geração de todas as palavras.

3. Desenvolvimento da Solução

Para resolução do problema descrito acima, utilizou-se a tecnologia CUDA codificada em linguagem C. Os códigos-fonte, bem como arquivos necessários para o processamento, podem ser obtidos em [1].

3.1. CUDA

CUDA [2] é uma plataforma que utiliza recursos de *software* e *hardware* para computação concorrente de alto desempenho de propósito geral, onde seu poder advém dos múltiplos núcleos da GPU NVIDIA. Trata-se de uma tecnologia muito eficiente, pois permite:

- Leitura paralela - o código pode ler de endereços arbitrários na memória;
- Memória compartilhada - CUDA expõe uma região de memória compartilhada rápida (16KB em tamanho) que podem ser compartilhados entre *threads*. Isso pode ser usado como um cache de usuário, permitindo maior largura de banda do que é possível utilizando textura *lookups*;
- Downloads mais rápidos e *readbacks* para a GPU;

- Suporte completo para operações de números inteiros e operações de *bitwise*.

Entretanto, possui certas limitações, tais como:

- A renderização de texturas não é suportada;
- As cópias realizadas entre uma memória e outra podem gerar algum problema na performance das aplicações;
- Ao contrário do OpenCL, o CUDA está disponível apenas para placas de vídeo fabricadas pela própria NVIDIA. Caso seja usado em outro tipo de placa, o CUDA funcionará corretamente, entretanto o desempenho será bem limitado.

3.2 Modelagem e Funcionamento do Programa

Para modelagem da solução, dividimos as tarefas em duas etapas:

1) Aplicação de filtragem do arquivo base de palavras;
2) Aplicação principal para geração de palavras aleatórias e busca em dicionário.

Para filtragem do arquivo base, todo o processamento foi executado usando apenas a CPU. A ideia do algoritmo é filtrar o arquivo *palavras.txt* tirando as palavras iguais e escrevê-las (uma palavra por linha) no arquivo intermediário *f.txt* (que será removido no final da execução), e em seguida, refiltrá-lo quebrando as palavras geradas (de tamanho real) em subpalavras de tamanho cinco e desprezando as de tamanho um. As palavras são armazenadas na estrutura de dados *Trie*, que permite uma rápida verificação (busca) e impede a duplicação.

Na aplicação principal, cada *thread* criada pelo CUDA gera palavras aleatórias formadas por uma letra, duas letras, três letras, quatro letras e cinco letras. Na medida em que vão sendo geradas, o algoritmo faz uma *Busca Binária* e as compara com o dicionário previamente carregado. Caso encontre a palavra, sinaliza uma marcação para garantir que quando reencontrá-la o contador principal não seja afetado. O algoritmo termina quando todas as palavras foram encontradas. A verificação no dicionário (*array* de estrutura contendo *string* para palavra e flag que simboliza se a mesma já foi encontrada) é feita dentro da thread CUDA, isto é, na GPU. Por outro lado, a verificação do contador de palavras encontradas com o total de palavras no dicionário é feita na própria CPU.

4. Testes e Resultados

Para obtenção dos resultados, foram realizados testes de desempenho comparando o número de blocos e número de threads. Nos gráficos abaixo, o eixo das abscissas representa a porcentagem de palavras encontradas e o eixo das ordenadas o tempo gasto em segundos.

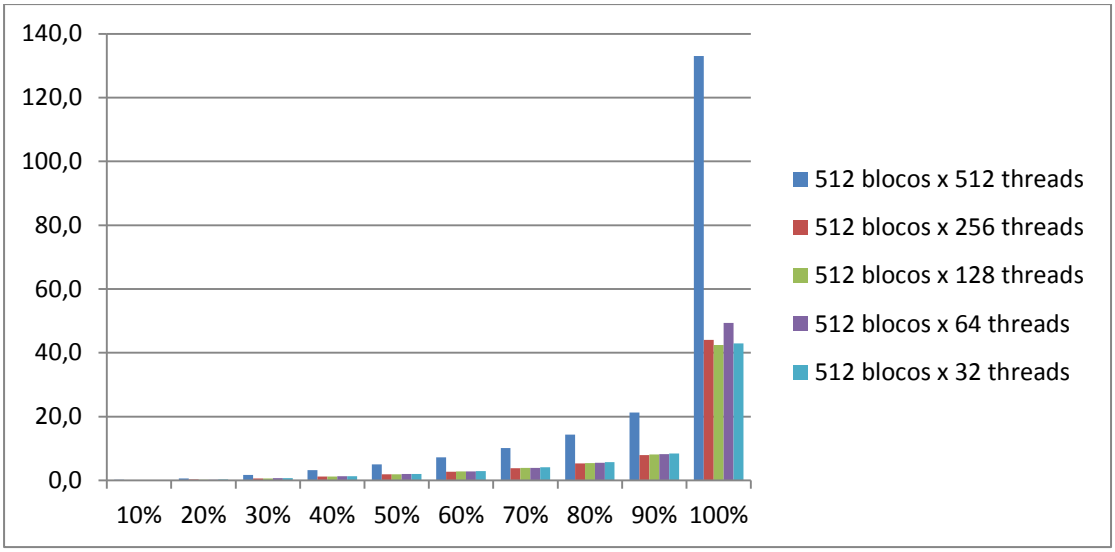

Figura 1 - Gráfico de 512 blocos

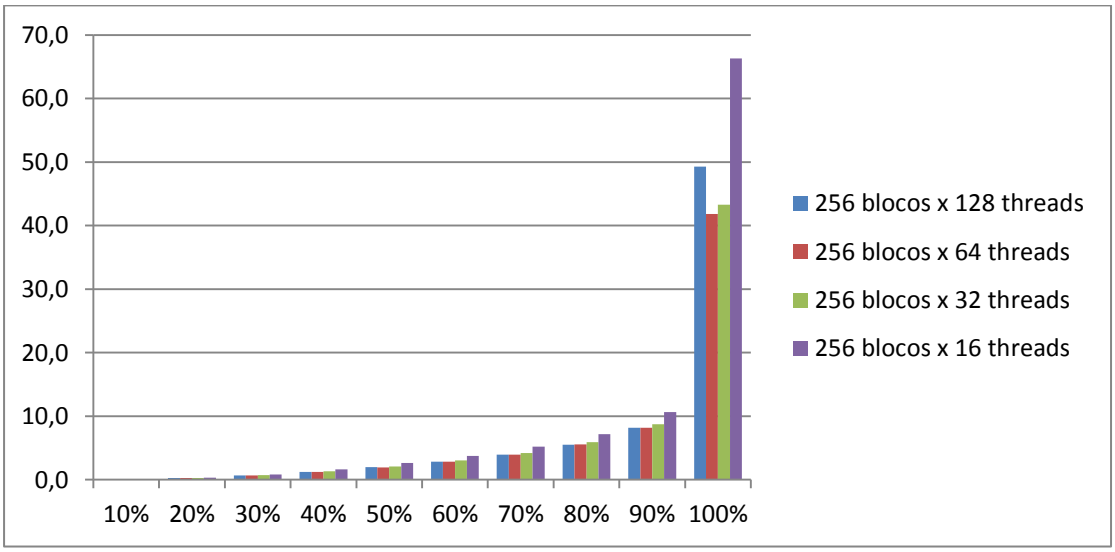

Figura 2 - Gráfico de 256 blocos

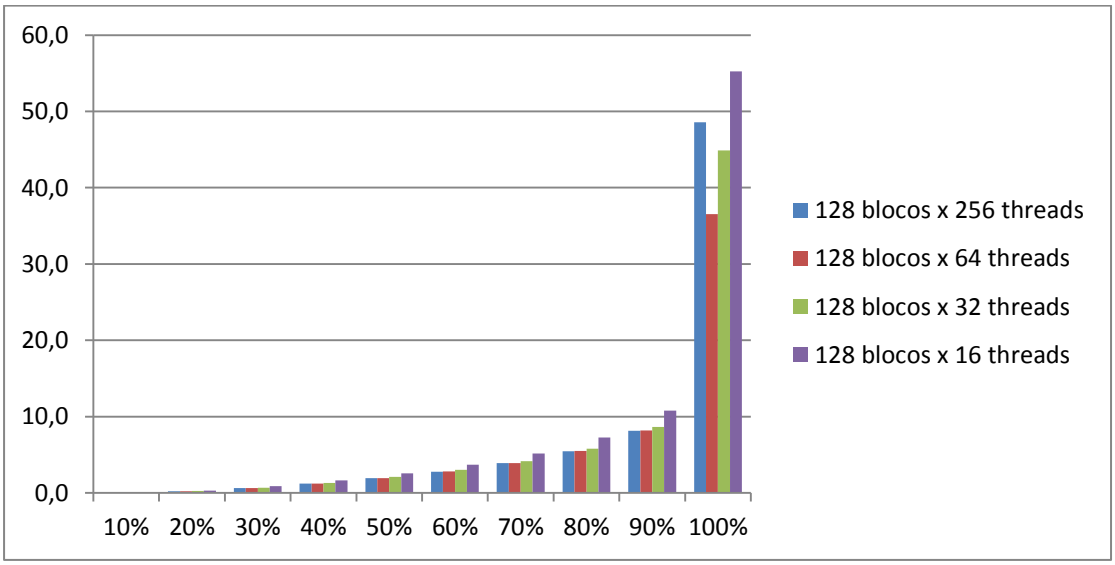

Figura 3 - Gráfico de 128 blocos

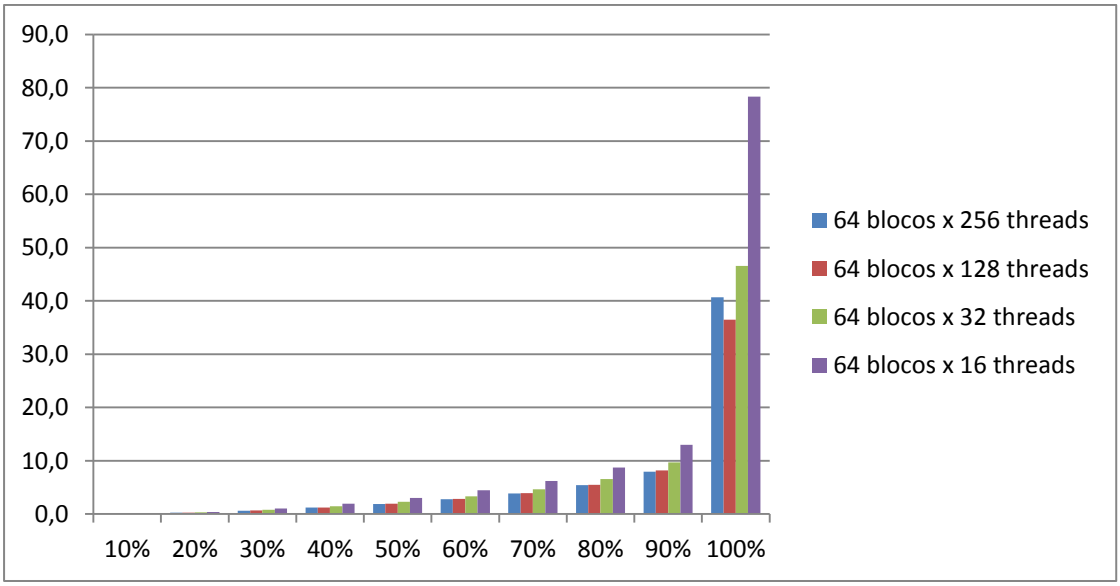

Figura 4 - Gráfico de 64 blocos

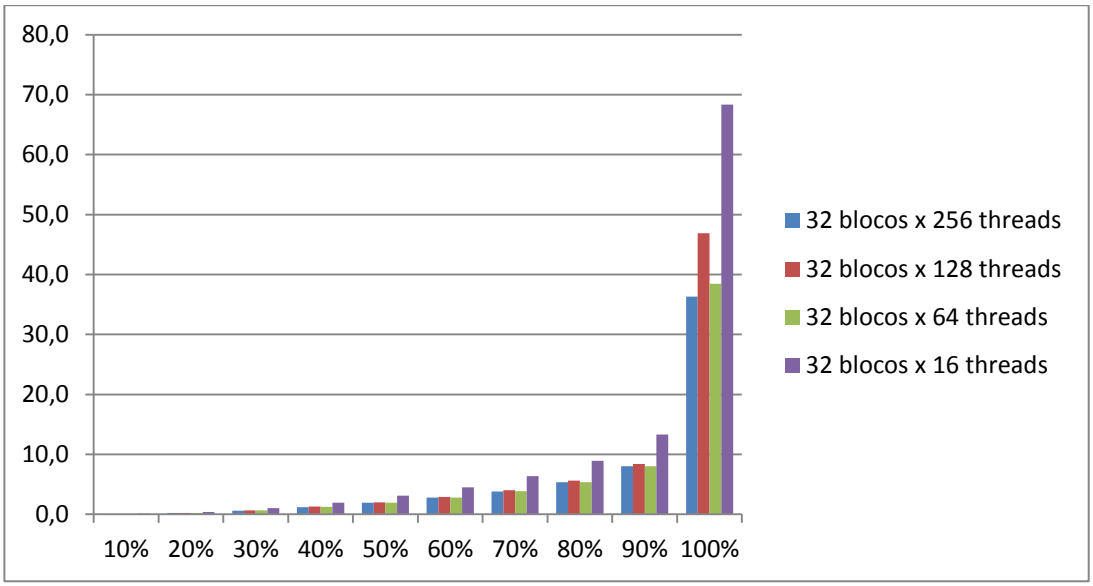

Figura 5 - Gráfico de 32 blocos

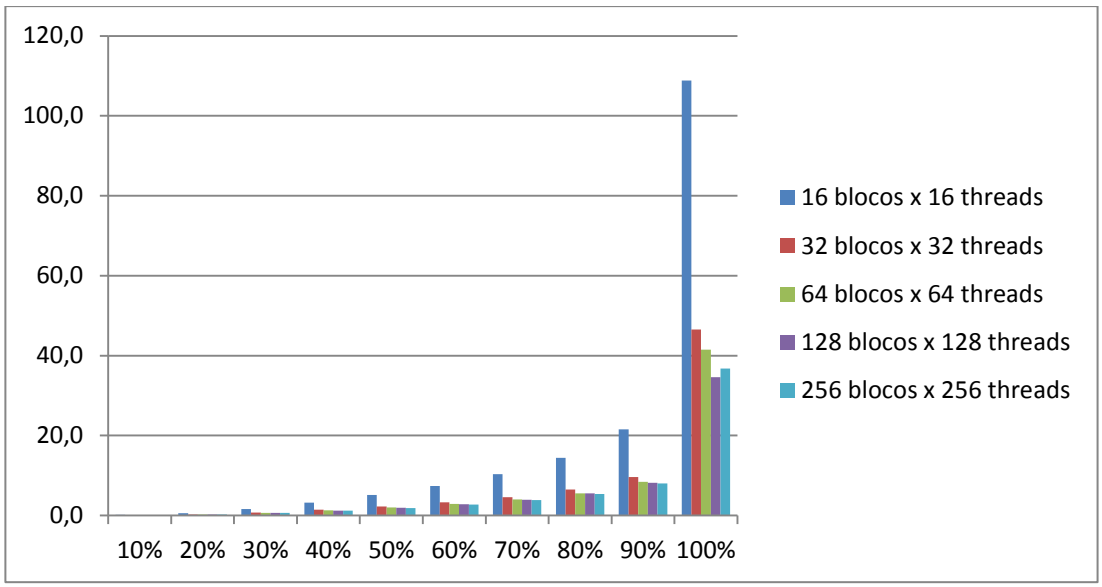

Figura 6 - Gráfico de blocos variáveis

Como se pode observar, os extremos de blocos e threads não apresentam bom desempenho. E os melhores balanços foram alcançados com 32 blocos x 256 threads, 128 blocos x 128 threads, 64 blocos x 128 threads, 128 blocos x 64 threads, 256 blocos x 64 threads e 512 blocos x 128 threads.

5. Conclusão

O projeto obteve resultados satisfatórios, pois inicialmente acreditava-se ser impossível gerar todas as palavras em um tempo razoável. Entretanto, após a filtragem do arquivo texto base, pudemos observar um ganho considerável no *throughput*.

Ao realizar os testes, notou-se que a quantidade de 512 threads se torna inviável por uma questão ligada ao processamento e não a limitações de hardware. Também se observou que a variação do número de blocos apresenta maior influencia sobre o tempo total do que o número de threads.

A temperatura interna da GPU apresenta uma drástica influência sobre o processamento. O fato deste não ser o foco do projeto fez-se uma breve análise que resultou num tempo de processamento para 512 blocos e 128 threads à 58°C e se obteve um tempo total de 33.385 segundos, com a mesma configuração mas à 75°C obteve-se 42.477 segundos.

6. Referências

[1] pc2012-grupo-9-turma-a. – Programação Concorrente – Google Codes. Disponível em: http://code.google.com/p/pc2012-grupo-9-turma-a/source/browse/#svn%2Ftrunk%2Fprojeto-final Acesso em: 30/06/2012.

[2] Aula 13 – Introdução - CUDA – disponível em: http://moodle.lasdpc.icmc.usp.br/pluginfile.php/1776/mod_folder/content/3/Aula-13-Introducao-Cuda.zip?forcedownload=1. Acesso em: 30/06/2012.