Análise Comparativa de Algoritmos NP-Completo Executados em CPU E GPU Utilizando CUDA

Elcio Arthur Cardoso, Rafael de Santiago

Curso de Ciência da Computação – Universidade do Vale do Itajaí (UNIVALI)
Caixa Postal 360 – CEP 88302-202 – Itajaí – SC – Brasil

{elcio_arthur, rsantiago}@univali.br

Abstract. *Research for greater computing power, always present in the computation that led to the creation of parallel architectures consisting of thousands of processing units, as occurs in the architectures of GPUs. In this context, this work presents a search for using the computational power of GPUs through CUDA architecture, aiming to solve computational problems found in the class NP-Complete, those that can take years to resolve. This paper shows three strategies to approach with their limitations and benefits. With these results, there is evidence that the architecture is efficient in solving these problems.*

Keywords: *CUDA. NP-Complete. Parallel Computing.*

Resumo. *A busca por maior poder computacional, sempre esteve presente na computação o que levou à criação de arquiteturas paralelas compostas por milhares de unidades de processamento, como ocorre nas arquiteturas das GPUs. Neste contexto, apresenta-se uma pesquisa para utilizar o poder computacional das GPUs, através da arquitetura CUDA, buscando resolver problemas computacionais, encontrados na classe NP-Completo, estes que podem levar anos para serem resolvidos. São apresentadas três estratégias de abordagem com suas limitações e benefícios. Com os resultados obtidos, há evidencias que a arquitetura é eficiente na solução destes problemas.*

Palavras-Chave: CUDA. NP-Completo. Computação Paralela.

1. Introdução

Segundo Tanembaum (2003), há uma demanda por computadores cada vez mais rápidos, demanda esta principalmente suprida devido a incrementos na velocidade de operação dos processadores, porém estas evoluções chegaram ao seu fim devido a limites físicos de evolução das CPUs (Central Processing Units), em alternativa para a evolução física das CPUs os fabricantes passaram a aumentar o número de processadores para suprir a demanda de desempenho em tempo de processamento.

De acordo com Sanders e Kandrot (2010) a fabricante NVIDIA lançou em 2006 a GeForce 8800 GTX, a primeira GPU construída com a arquitetura CUDA (Compute Unified Device Architecture), que apresentou soluções a várias limitações que impediam que as placas anteriores fossem legitimamente usadas para computação de

proposito geral, consequentemente apresentando alta performance para aplicações heterogêneas com a utilização das unidades central e gráfica de processamento.

De acordo com Sipser (2007), a pesquisa por desempenho não é realizada somente no Hardware. Existem áreas da Ciência da Computação que estudam a questão em algoritmos como, por exemplo, a Teoria da Complexidade. Esta área busca investigar e compreender o porquê de algoritmos terem características de consumo de recursos (geralmente espaço em memória e tempo). Uma das maiores contribuições que os pesquisadores da área já realizaram é a definição de um esquema de classificação de algoritmos, onde se é possível posicioná-los de acordo com seu grau de complexidade.

Sipser (2007) cita que dentre estas classes, a classe P, formada pelos problemas que podem ser resolvidos em tempo polinomial com algoritmo determinístico. Outro exemplo importante é a classe NP, que é composta por problemas que podem ser verificados em tempo polinomial e que podem ser resolvidos por algoritmo não-determinístico em tempo polinomial, incluindo então, diversos problemas que são resolvidos em tempo exponencial. Para a Ciência da Computação teórica e matemática contemporânea a questão de "se P = NP?" é um dos maiores problemas não resolvidos. Caso essas classes sejam iguais, qualquer problema polinomialmente verificável seria polinomialmente “decidível”.

Garey e Johnson (1979) afirmam que os algoritmos da classe NP-Completo são grandes motivadores da computação de alto desempenho, neste sentido foi analisado o desempenho de dois algoritmos exatos para problemas NP-Completo ao serem executados sequencialmente em uma CPU e paralelamente em uma GPU NVIDIA utilizando a arquitetura CUDA, unindo dois temas: problemas computacionalmente intratáveis e uma das tecnologias responsáveis por maior desempenho computacional.

2. Teoria da Complexidade

Classes de complexidade são agrupamentos de problemas computacionais que podem ser decididos utilizando o mesmo recurso computacional afirmam Aurora e Barak (2006), as que têm principal relação com este são: P, NP e NP-Completo.

Segundo Ziviani (2007), os algoritmos podem ser divididos em duas classes, determinísticos e não determinísticos, para o primeiro o resultado de cada operação é definido de maneira única, já o segundo é um dispositivo teórico capaz de realizar a execução em múltiplas configurações consumindo o tempo de apenas uma.

A classe P (*polynomial time*) é composta pelos problemas que podem ser resolvidos em tempo polinomial por uma máquina de Turing determinística, ou seja, são problemas que podem ser solucionados por um computador em tempo aceitável, afirma Sipser (2007). Um exemplo de problema inserido na classe P é o problema do Caminho (CAM).

De acordo com Sipser (2007) e Ziviani (2007), problemas que podem ser verificados em tempo polinomial e que podem ser resolvidos por um algoritmo não determinístico em tempo polinomial estão na classe NP (*Non-Deterministic Polynomial Time*). Um exemplo de problema que compõe a classe NP é o do Caminho Hamiltoniano (CAMHAM).

Segundo Garey e Johnson (1979), o teorema de Cook-Levin comprova que há um grupo de problemas em NP que se houver um algoritmo polinomial capaz de resolver qualquer um desses problemas, então todos os problemas da classe NP seriam solúveis em tempo polinomial, questão P versus NP, esses problemas pertencem à classe NP-Completo e são o alvo dos estudos deste trabalho.

Garey e Johnson (1979) listam uma vasta lista de problemas que compõem a classe NP-Completo, neste trabalho foram desenvolvidos desses: Clique Máximo do Grafo e Cobertura Mínima de Vértices. O primeiro tem como objetivo localizar dentro de um grafo o maior conjunto de vértices sendo que todos possuam conexão mutua com os demais vértices do conjunto. O segundo tem como objetivo localizar dentro de um grafo o menor subconjunto de vértices na qual todas as arestas tenha ligação em ao menos um desses vértices.

3. Computação Paralela

A demanda por computadores cada vez mais rápidos nos mais diversos contextos comerciais e científicos existe desde o Eniac, até hoje, com máquinas um milhão de vezes mais rápidas afirma Tanembaum (2003).

Tanembaum (2003) afirma ainda, que a partir de 1980, o principal método de aumento de desempenho nas CPUs ocorreu devido a incrementos na frequência de operação. Seguindo com essa evolução, barreiras foram encontradas no aumento da frequência nas CPUs, tais como: alto consumo de energia, restrições de calor e também pelo limite físico de fabricar resistores menores do que os atuais. Com estas limitações, pesquisadores e fabricantes procuraram por outros modelos para se ganhar desempenho.

Sanders e Kandrot (2010) afirmam que como alternativa aos limites físicos de aumento de desempenho das CPUs, o aumento no número de processadores como foi adotado como alternativa para suprir a demanda, o que tornou comum o uso de supercomputadores multiprocessados com até milhares de processadores trabalhando em conjunto.

Harris (2005) e Viana (2011) apontam que juntamente com a evolução das CPUs, as GPUs (Graphics Processing Unit) também passaram por uma grande evolução, principalmente focadas em uma arquitetura com o maior número de circuitos de processamento do que para cache, o que levou as GPUs a apresentarem um crescimento superior a duas vezes ao ano e mil vezes em uma década.

Segundo Ikeda (2011) em 1970 surgiram os primeiros aceleradores gráficos 2D, de 1995 a 1998 os 3D que tiveram um papel importante no desenvolvimento das GPUs. Com o passar dos anos os aceleradores gráficos deixaram de ser exclusivamente para processamento de imagens dando lugar as GPUs programáveis, passando a ser possível o aproveitamento do enorme poder computacional presente neste hardware paralelo.

Em 2007 foi lançada pela fabricante NVIDIA a GeForce 8800 GTX, segundo Ikeda (2011), a nova arquitetura CUDA presente nesta GPU oferecia suporte para linguagens de programação já conhecidas como o C e era totalmente focada a computação paralela de alto desempenho, deixando para trás as principais dificuldades da utilização do *hardware* gráfico para uso de programação de propósito geral (GPGPU).

Segundo a NVIDIA (2009), a arquitetura CUDA permite que *kernels*, funções executas na GPU, ao serem executados possam ser configurados para obterem o melhor desempenho do *hardware* de acordo com cada problema, o mesmo ocorre com a utilização das diferentes memórias disponíveis na arquitetura.

4. Desenvolvimento

Os algoritmos foram implementados para serem executados na CPU, também foram definidas estratégias de abordagem dos problemas e feitas às implementações para GPU utilizando CUDA. Com isso foi possível executar os testes e analisar os resultados.

4.1. Algoritmos NP-Completo para CPU

Os algoritmos Clique Máximo foi desenvolvido para a CPU seguindo a logica definida por Kreher e Stinson (1999), na qual a função recursiva MaxClique é responsável por percorrer o grafo em busca da solução. Os resultados são obtidos através do processo de *backtraking* na qual todas as possibilidades de solução são testadas até que o Clique Máximo seja encontrado.

Assim como o Clique Máximo, o algoritmo Cobertura Mínima de Vertices também foi implementado através da função CobMin seguindo a mesma lógica de implementação.

Por analisarem todas as possibilidades, os algoritmos ao executarem podem ser classificados pela complexidade pessimista, como exponenciais representado pela função de limite superior $O(2^n)$.

4.2. Algoritmos NP-Completo para GPU

Inicialmente foi utilizada uma estratégia similar à apresentada por Blellock (1990) para paralelização das funções similares e que dependem dos mesmos dados para serem executadas, abordado assim uma árvore com a complexidade próxima a da step complexity, na qual cada nó presente no mesmo nível da árvore é executado paralelamente por um diferente kernel.

No decorrer do uso desta estratégia problemas relacionados ao uso memória foram encontrados, nos quais as estruturas dos dados responsáveis por armazenar o maior clique ou maior cobertura já encontrada e o caminho que já havia sido percorrido não podiam mais ser compartilhadas, mas sim exclusivas de cada kernel, resultando assim, não só na troca do uso exponencial de tempo para o uso exponencial de processamento, mas também para o consumo exponencial de memória, representado pela equação 2(n-1), na qual n = número de vértices e que tem como resultado o número de caminhos possíveis de serem percorridos, ou seja, a quantidade de conjuntos de memória para ser possível gerar o resultado final é exponencial. Um exemplo deste consumo seria para um grafo formado por 40 vértices, na qual seriam necessários cerca de 1TB de memória para guardar os possíveis caminhos.

4.3. Soluções alternativas para GPU

Devido ao fato da solução inicial proposta neste trabalho não ter sido possível, o foco do mesmo se tornou em buscar abordagens alternativas na qual os problemas NP-Completo fossem capaz de tirar proveito da arquitetura CUDA.

4.3.1. Solução alternativa – Paralelizando operações

Por ser comum em problemas NP-Completo o uso exaustivo de operações de união e intersecção de vetores, foi implementado, como solução alternativa à estratégia inicial, um algoritmo em CUDA seguindo a mesma lógica dos algoritmos desenvolvidos para a CPU, porém operações com os elementos em uma lista puderam ser paralelizadas para executarem na mesma unidade de tempo, assim como outras otimizações, como, por exemplo, uma verificação de elementos vazios na lista, na qual todos os elementos também puderam ser verificados em uma única unidade de tempo.

Mesmo com as otimizações feitas nas operações com os elementos dos vetores e matrizes, o desempenho obteve melhoras apenas em um dos ambientes, entretanto os mesmos não foram significativos quando comparados com a CPU. Este resultado ocorreu devido ao fato desta implementação exigir muitas leituras e gravações na memória global da GPU, que é considerada lenta, levando assim a resultados que comprovam que uma lógica para CPU nem sempre obtém bons resultados quando levada para a GPU, mesmo porque não houve alteração na função de limite superior de complexidade. Esta comparação entre CPU e GPU paralelizando operações pode ser acompanhada mais adiante na seção de resultados.

4.3.2. Solução alternativa – Subgrafos

Como os resultados obtidos com a primeira solução alternativa não foram satisfatórios uma nova estratégia foi buscada pensando em dividir o grafo e processá-las em paralelo, respeitando as limitações apresentadas por CUDA, na solução anterior, e visando explorar onde a arquitetura se mostrou mais eficiente.

Para o Clique Máximo do grafo foi adotada a estratégia de dividir o grafo testado em n subgrafos, onde n é o número de vértices do grafo, ou seja, n outros grafos menores conforme ilustra a Figura 1.

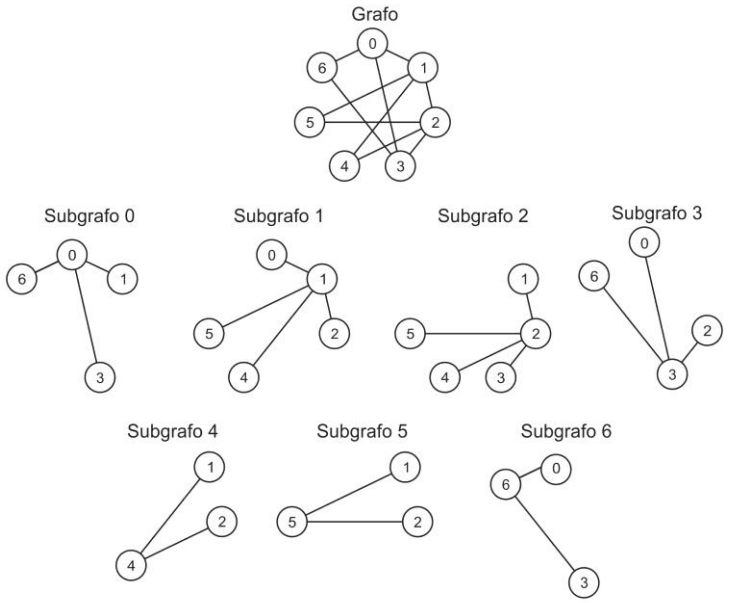

Figura 1. Divisão do grafo em n subgrafos

Para cada subgrafo foi disparado um bloco, no total de n, número de vértices, blocos executados em paralelo, capazes de resolver o Clique Máximo conectado ao vértice inicial do subgrafo, por exemplo, no subgrafo tratado pelo bloco zero o vértice zero é o vértice inicial e o resultado da execução deste bloco é dado pelo maior clique envolvendo o vértice zero, o mesmo o corre para os vértices um, dois, três, até n. No final o tamanho de cada clique é comparado a fim de encontrar o maior dentre eles.

Esta solução acaba por executar cálculos repetidos, como por exemplo, o maior clique é encontrado p vezes, na qual p é a quantidade de vértices do maior clique, entretanto estes cálculos repetidos são executados paralelamente, não impactando assim consideravelmente no tempo total da execução.

5. Teste e Resultados

Após o desenvolvimento dos algoritmos foram feitos os testes de execução dos mesmos, na qual foram selecionados três diferentes ambientes visando garantir desempenho e confiabilidade dos resultados, além disso, foram selecionados grafos já conhecidos para garantir que os resultados dos mesmos estivessem corretos. Feitas as execuções os tempos das mesmas foram verificados e comparados.

5.1. Ambiente

Os testes realizados neste trabalho foram divididos em três ambientes:

- CPU: notebook composto por uma CPU Intel Core i3 M370 de 2,40GHz, 3GB de memória RAM DDR3;
- GPU 1: computador composto por uma CPU Intel Pentium 4 de 2,26GHz, 2GB de memória RAM DDR2 e uma GPU NVIDIA GeForce GT 640 possuindo 384 núcleos CUDA, interface de memória 128-bit e 2GB de memória DDR3; e
- GPU 2: computador composto por uma CPU Intel Core i5 de 3,3GHz, 12GB de memória RAM DDR3 e uma GPU NVIDIA GeForce GTX 670 possuindo 1344 núcleos CUDA, interface de memória 256-bit e 2GB de memória GDDR5.

Os algoritmos em CUDA foram compilados utilizando o NVIDIA CUDA compiler driver (nvcc) versão 4.2 e foram utilizados os parâmetros "-gencode=arch=compute_30,code=sm_30,compute_30" para utilizar as operações disponíveis nas GPUs *compute capability* 3.0 e obter melhor desempenho das GPUs desta versão da arquitetura CUDA, uma vez que ambas utilizadas nos testes fazem parte desta.

5.2. Grafos

Os grafos utilizados para os testes são listados na Tabela 1 e foram selecionados dentre os grafos disponibilizados pela DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), um grupo norte americano formado por empresas e universidades para estudo envolvendo aplicações práticas da matemática discreta e teoria da computação, em seus desafios que buscam determinar o desempenho de alguns algoritmos de seu interesse segundo DIMACS (2012).

Tabela 1. Grafos

	Vértices	Arestas	Vértices no Clique Máximo	Vértices na Cobertura
Johnson 8-2-4	28	210	4	24
Hamming6-4	64	704	4	60
c-fat200-1	200	1534	12	188

5.3. Resultados

Os algoritmos foram executados e o tempo de execução da função principal foi coletado, para garantir que os resultados sejam os mais precisos possíveis, o resultado é dado pela média do tempo de cinco execuções em cada ambiente, na qual o mesmo estava dedicado exclusivamente aos algoritmos descritos por este documento, reduzindo assim interferências pelo sistema operacional e de outros aplicativos que pudessem estar executando no mesmo momento.

Os tempos resultantes das execuções do Clique Máximo do grafo podem ser vistos e comparados na Figura 2, que exibe os resultados das execuções em CPU, de forma sequencial e dos algoritmos Paralelizando operações e Subgrafos nos dois ambientes, já descritos, utilizando GPU.

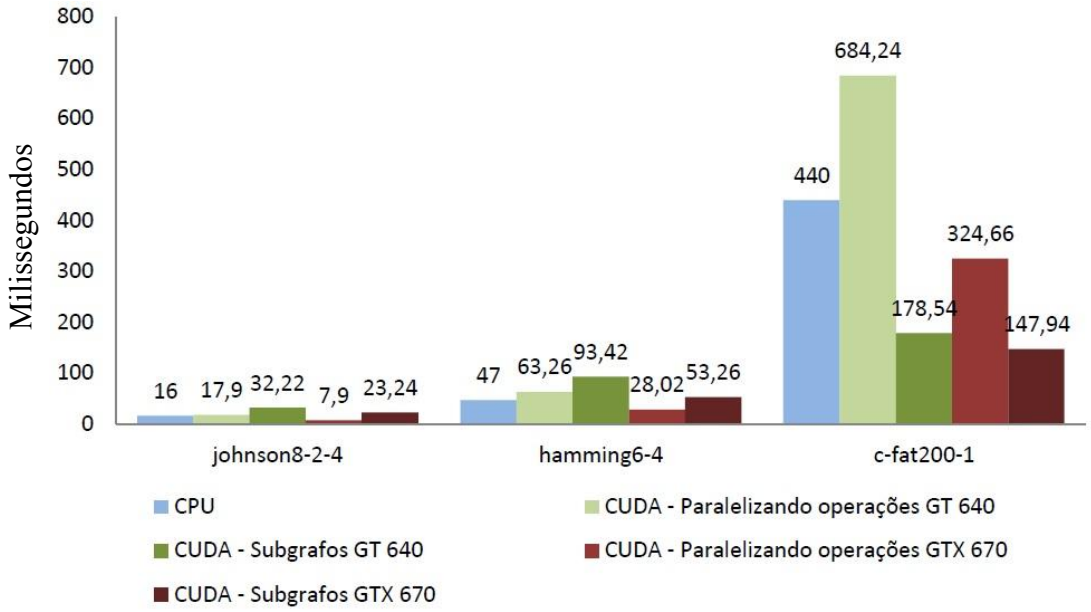

Figura 2. Resultados da execução do clique do grafo

Com tempos resultantes das execuções do Clique Máximo do grafo é possível observar que a CPU apresenta vantagens sobre a GPU 1 quando os grafos são menores, ou seja, quando se tem pouco processamento em relação a quantidade dos dados a serem processados, o que não ocorre com a GPU 2 pois a mesma é bem mais eficiente neste tipo de operação, já as GPUs apresentam vantagem quando os grafos se tornam maiores, essa vantagem é dada uma vez que a quantidade de dados aumentam pouco e seu tempo de alocação na memória gráfica cresce proporcionalmente, entretanto a quantidade de processamento executado sobre esses dados cresce exponencialmente.

Em resumo, nas GPUs, o algoritmo executando grafos menores passa mais tempo preparando os dados do que os executando, já nos grafos maiores, o tempo para preparar os dados deixa de ser significativo devido ao tempo de processamento se tornar sobressalente e este crescimento tende a ser mais notável na medida em que os grafos forem crescendo.

6. Conclusão

Com este trabalho pôde-se concluir que quando se troca o tempo exponencial em processamento exponencial também se torna necessário o uso de memória exponencial, na qual esta alternativa não pode ser aplicada, sendo assim necessária a busca por soluções alternativas.

Problemas NP-Completos são resolvidos em CPU através de árvore de *backtracking* juntamente com estratégias de buscas exaustiva (em profundidade) procurando assim exaurir todas as possibilidades até encontrar o melhor conjunto de configurações em relação à resposta ao problema. Em CUDA boas estratégias devem se basear em dividir o problema em etapas evitando a forma sequencial de solução (*backtracking* junto com busca exaustiva). Para CUDA deve-se dividir o problema em etapas de acordo com a estratégia "divisão por conquista" buscando paralelizar as operações e evitando consumo excessivo de memória, uma vez que soluções paralelizáveis possuem memórias de acesso rápido muito restrito.

Com isso, verificando os testes e comparações realizadas, em busca do objetivo geral deste trabalho, comprovou-se assim a eficiência de CUDA com os problemas NP-Completo uma vez que os mesmos sejam abordados de forma a utilizar os benefícios e respeitar as limitações da arquitetura.

Referências Bibliográficas

Arora, S. e Barak, B. (2009), Computational Complexity: a modern approach. 1ª edição, Cambridge University Press, ISBN 978-05-2142-426-4.

Blelloch, G. (1990). "Vector Models for Data-Parallel Computing", MIT Press, ISBN 978-02-6202-313-9.

DIMACS. (2012). "DIMACS Implementation Challenges". http://www.ufpi.br/subsiteFiles/ercemapi/arquivos/files/minicurso/mc1.pdf, outubro 2012.

Garey, M. R. e Johnson, D. S. (1979), Computers and Intractability: a guide to the theory of NP-Completeness, Bell Telephone Laboratories Inc, ISBN 0-7167-1044-7.

Harris, M. (2005), "GPGPU: General-Purpose Computation on GPU ". ftp://216.151.177.77/developer/presentations/2005/GDC/OpenGL_Day/OpenGL_GPGPU.pdf, março, 2012.

Ikeda, P. A. (2011). "Um estudo do uso eficiente de programas em placas gráficas".

Kreher, D. L., Stinson, D. R. (1999). "Combinatorial Algorithms: Generation, Enumeration and Search, CRC Press, ISBN 978-08-4933-988-2.

NVIDIA. (2009), "NVIDIA CUDA Architecture: Introduction & Overview". http://developer.download.nvidia.com/compute/cuda/docs/CUDA_Architecture_Overview.pdf, março, 2012.

Sanders, J. e Kandrot, E. (2011). CUDA by example: na introduction to General-Purpose GPU Programming, Pearson Education, ISBN: 978-0-13-138768-3.

Sipser, M. (2007). Introdução à Teoria da Computação, 2ª edição, Cengage Learning, ISBN: 978-85-221-0499-4.

Tanenbaum, A. S. (2003). Sistemas operacionais modernos, 2ª edição, Pearson Prentice Hall, ISBN: 8587918575.

Viana, J. R. M. (2011), "Programação em GPU: Passado, presente e futuro". http://www.ufpi.br/subsiteFiles/ercemapi/arquivos/files/minicurso/mc1.pdf, março, 2012.

Ziviani, N. (2007). Projeto de algoritmos, Thomson, 2007.