IMPLEMENTAÇÕES PARALELAS PARA O PROBLEMA DA MOCHILA MULTIDIMENSIONAL USANDO ALGORITMOS GENÉTICOS

Bianca de Almeida Dantas
Faculdade de Computação – Universidade Federal de Mato Grosso do Sul
CEP: 79070-900 - Caixa Postal 549 Campo Grande/MS – Brasil
bianca@facom.ufms.br

Edson Norberto Cáceres[1]
Faculdade de Computação – Universidade Federal de Mato Grosso do Sul
CEP: 79070-900 - Caixa Postal 549 Campo Grande/MS – Brasil
edson@facom.ufms.br

RESUMO

O problema da mochila é um problema importante na área de otimização combinatória e tem sido objeto de muitas pesquisas nas últimas décadas. O problema possui diversas variantes e a obtenção de uma solução exata não é uma tarefa fácil, o que motiva a busca por soluções alternativas. Dentre tais alternativas, as heurísticas merecem destaque, especialmente os algoritmos genéticos que em várias situações obtêm bons resultados para o problema. Com a grande disponibilidade de recursos computacionais de alto desempenho, tem-se o desafio de utilizar esse poder computacional para obter soluções ainda melhores. Os algoritmos genéticos demonstram um grande potencial de paralelização e isso possibilita sua utilização em problemas maiores. Neste trabalho são propostas implementações paralelas para o problema da mochila multidimensional usando algoritmos genéticos e, então, são comparados os resultados obtidos visando definir as características dos problemas para os quais o uso de técnicas de paralelização pode ser vantajoso.

PALAVRAS CHAVE. Algoritmos genéticos, problema da mochila multidimensional, programação paralela.

Área Principal: Algoritmos Genéticos.

ABSTRACT

The knapsack problem is an important problem in the combinatorial optimization area and has been object of many researches in the last decades. The problem has a great number of variants and to achieve an exact solution is not an easy task and this motivates the search for alternative solutions. Among these alternatives, heuristic techniques deserve special attention, mainly genetic algorithms which usually achieve good solutions to the problem. With the wide availability of high performance computational resources, we are challenged to use this computational power in order to achieve even better solutions. Genetic algorithms show great parallelization potential and this enables them to be used on larger problems. In this work we propose parallel programs for the multidimensional knapsack problem using genetic algorithms and then compare the obtained results to determine the features of problems which parallelization techniques can be successfully applied into.

KEYWORDS. Genetic algorithms. Multidimensional knapsack problem. Parallel programming.

Main Area: Genetic algorithms.

[1]Parcialmente apoiado pelo CNPq.

1. Introdução

O problema da mochila possui uma ampla gama de aplicações. Sua variante 0-1 possui basicamente duas entradas, a primeira é um conjunto de itens (com seus valores e pesos associados) e a segunda a capacidade da mochila; sendo que há duas possibilidades para cada item: ele deve ou não ser levado. O objetivo é escolher os itens de forma a maximizar o valor a ser carregado.

Apesar de sua relativa simplicidade, esse problema é $\mathcal{NP}$-completo, ou seja, não é conhecida a existência de um algoritmo polinomial para sua solução exata. Existe um algoritmo de programação dinâmica simples para solução desse problema com complexidade $O(nW)$ que, pelo fato de depender tanto do número de itens quanto da capacidade da mochila, é denominado pseudopolinomial. Esse algoritmo possui versões paralelas com bom *speedup* (Cáceres, 2005).

Quando o número de restrições impostas ao problema da mochila aumenta, tem-se o problema da mochila multidimensional, no qual as alternativas de solução tradicional não podem ser eficientemente aplicadas. Essa situação motiva a busca por alternativas para obter uma solução para o problema, ainda que aproximada, dado que a obtenção da solução exata pode ser proibitiva devido ao seu alto custo de tempo de execução. Nesse contexto, a utilização de estratégias como algoritmos de aproximação e heurísticas torna-se atrativa. Além disso, em função da complexidade e dos tempos é importante a obtenção de algoritmos paralelos competitivos para esse problema.

Diversos algoritmos paralelos foram propostos para resolver o problema da mochila multidimensional, os quais diferem pelas técnicas utilizadas e pela qualidade das soluções obtidas. Burns (Burns, 1993) apresentou implementações paralelas do algoritmo enumerativo ordenado multidimensional (*multidimensional ordered enumerative algorithm* - MOEA), proposto por Lebedev (Lebedev, 1968), que visava melhorar a eficiência da versão sequencial do algoritmo utilizando múltiplos processadores. No ano seguinte, Burns e Friedman (Burns, 1994) propuseram outros algoritmos paralelos para o problema utilizando redes de *transputer*. Em 1997, Niar e Freville (Niar, 1997) propuseram uma implementação paralela de uma metaheurística baseada na busca tabu, eles mostraram que a utilização do paralelismo permitiu redução do tempo para obtenção de uma solução e, ainda, possibilitou que a configuração de alguns parâmetros da busca tabu fosse realizada de maneira automática e dinâmica; essa característica mostrou um balanceamento entre a intensificação e a diversificação do processo. Posadas et al. (Posadas, 2010) implementaram uma versão paralela de algoritmo genético para resolver, em especial, instâncias de tamanho mediano. Nesse trabalho, os autores usaram as bibliotecas GALib e OOMPI que implementam, respectivamente, funcionalidades de algoritmos genéticos e comunicação. Fingler (Fingler, 2013) propõe uma implementação em CUDA utilizando colônias de formigas, baseada na proposta por Solnon e Bridge (Solnon, 2006), que obteve boas soluções em pouco tempo de execução. Liu e Lv (Liu, 2013) propuseram um algoritmo utilizando otimização por colônia de formigas baseado no modelo de programação *map-reduce* para sua paralelização. O algoritmo desenvolvido foi utilizado para resolver instâncias grandes do problema usando computação em nuvem.

Neste trabalho explorou-se a utilização de algoritmos genéticos, com uma implementação sequencial e duas paralelas usando MPI e CUDA, respectivamente. Os resultados obtidos são competitivos com os apresentados na literatura.

O restante do trabalho está organizado conforme segue. A seção 2 abrange a fundamentação teórica do problema da mochila multidimensional, de algoritmos genéticos e suas possibilidades de paralelização. A seção 3 apresenta detalhes das implementações realizadas. A seção 4 apresenta os resultados obtidos com as execuções dos programas. Ao final são apresentadas as considerações finais sobre o trabalho.

2. Fundamentação Teórica

2.1. Problema da Mochila Multidimensional

Existem diversas variantes do problema da mochila na literatura as quais se diferenciam, geralmente, pelo número de restrições impostas à solução ou por lidar com itens que podem ou não ser particionados. Este artigo se concentra no estudo do problema da mochila multidimensional

que consiste em, dado um conjunto de n diferentes itens, cada qual com um valor associado e m diferentes recursos e suas capacidades, decidir quais dos itens podem ser levados na mochila visando maximizar o valor carregado sem extrapolar a capacidade de nenhum dos recursos.

Formalmente, pode-se formular o problema de acordo com a equação 1 (Chu, 1998):

$$\max\{\sum_{j=1}^{n} v_j x_j\}, j = 1, ..., n \quad (1)$$

tal que

$$\sum_{j=1}^{n} r_{ij} x_j \leq b_i, i = 1, ..., m \quad (2)$$

onde n é o número de itens, m é o número de recursos, $V = \{v_1, v_2, ..., v_n\}$ é o vetor de valores dos itens, $B = \{b_1, b_2, ..., b_m\}$ é o vetor de capacidades dos recursos e $X = \{x_1, x_2, ..., x_n\}$ é o vetor de solução, no qual cada elemento pode possuir os valores 1 ou 0, equivalendo, respectivamente, à sua presença ou não na solução. A matriz R, por sua vez, representa o quanto cada item usa de cada restrição. A solução do problema consiste em encontrar um vetor solução X que respeite as capacidades de todos os recursos.

Uma alternativa para obter uma solução ótima é aplicar programação dinâmica considerando todas as m restrições. Contudo, a complexidade dessa alternativa é significantemente maior do que o da mochila binária, uma vez que será necessário construir uma matriz de programação dinâmica de $m + 1$ dimensões. Assim, os custos do uso da programação dinâmica para instâncias grandes do problema a tornam uma abordagem proibitiva; de fato, a obtenção de uma solução exata para a mochila multidimensional é um problema $\mathcal{NP}$-difícil e, em geral, não é aconselhada.

Como a obtenção da solução exata para o problema é um processo custoso, o uso de técnicas que buscam por soluções aproximadas tem sido intensivamente pesquisado como, por exemplo, algoritmos de aproximação e heurísticas como, por exemplo, estão os algoritmos genéticos.

2.2. Algoritmos Genéticos

Um algoritmo genético (AG) é uma técnica de busca e otimização baseada em princípios da genética e da seleção natural (Haupt, 2004), em que os indivíduos mais fortes de uma população tem chances maiores de transferir seus genes para as próximas gerações. A ideia principal reside na existência de uma população composta por diversos indivíduos que evoluem de acordo com regras de seleção específicas. A evolução ocorre até que os indivíduos atinjam um estado que maximize a sua adequação ao ambiente ou, ainda, até que atinjam um determinado número de gerações. A técnica foi desenvolvida ao longo das décadas de 1960 e 1970 por John Holland e descrita em seu livro de 1975 (Holland, 1975). Algumas das principais vantagens apresentadas pelos AGs com relação a outras técnicas de otimização são (Haupt, 2004):

- Permite a otimização com variáveis discretas e com variáveis contínuas;
- Permite lidar com problemas que têm um grande número de variáveis;
- Resultados são apresentados como uma população de soluções e não como uma solução única;
- Facilmente adaptável a ambientes paralelos.

De uma maneira simples, pode-se representar possíveis soluções para um problema como cadeias de bits e, a partir dessa representação, calcular a adequação de cada solução comparada às demais – o que é realizado com a utilização de uma função de *fitness*. Cada solução pode ser vista como um indivíduo e cada bit representa um de seus genes. Com base no valor calculado para o *fitness* de cada indivíduo, selecionam-se os "pais" dos indivíduos da próxima geração; esses passos

Algoritmo 1: Algoritmo Genético Simples.

```
1 Gera a população inicial;
2 Calcula o fitness dos indivíduos;
3 repita
4     Seleciona pais na população atual;
5     Reprodução;
6     Avalia o fitness da cada filho;
7     Substitui alguns indivíduos ou toda a população pelos novos indivíduos;
8 até uma solução satisfatória seja encontrada;
```

são repetidos por um determinado número de gerações ou até que uma determinada condição seja atingida. Os passos de um algoritmo genético são apresentadas no algoritmo 1 (Chu, 1998).

Um AG é caracterizado essencialmente por cinco componentes fundamentais: representação genética, população inicial, função de avaliação, operadores genéticos e parâmetros que controlam, por exemplo, o tamanho da população, o número de genes de cada indivíduo, a probabilidade de aplicação dos operadores genéticos, entre outros.

2.3. Algoritmos Genéticos Paralelos

Os algoritmos genéticos podem ser paralelizados considerando o modelo de programação paralela que se pretende adotar, destacando-se três alternativas (Cantu, 1998): população única global utilizando o paradigma mestre-discípulo, população única com granularidade fina e múltiplas populações com granularidade grossa, além de um modelo que pode misturar as alternativas anteriores (modelo híbrido hierárquico). Neste trabalho foram utilizados os dois últimos modelos.

Algoritmos genéticos de granularidade fina trabalham com uma única população; o que define como serão feitas as interações entre os indivíduos é a distribuição espacial dos elementos de processamento. Um indivíduo pode ser comparado e associado somente a um de seus adjacentes para realizar a seleção e a reprodução; entretanto, como os elementos de processamento formam uma grade conexa de interligação, as boas soluções irão se propagar por toda a população.

O modelo utilizando granularidade grossa considera a existência de um conjunto de subpopulações relativamente grandes distribuídas entre os elementos de processamento. Nesse tipo de paralelização, é essencial que exista uma maneira de evitar que as subpopulações evoluam de maneira totalmente isoladas das demais, como se fossem apenas diversas instâncias do AG sequencial; para evitar esse comportamento, utiliza-se o mecanismo de migração.

A migração consiste em trocar indivíduos entre as subpopulações após uma determinada quantidade de gerações. Surgem com isso novos parâmetros que devem ser definidos antes da execução de um AG com granularidade grossa tais como: tamanho das subpopulações, frequência das migrações, escolha dos elementos a serem migrados e quais devem ser substituídos com a chegada dos novos elementos, entre outras.

3. Implementação

Para verificar a viabilidade da utilização de algoritmos genéticos à solução do problema da mochila multidimensional e comparar as diferentes alternativas de paralelização, foram implementadas uma versão sequencial do algoritmo e as seguintes abordagens paralelas:

- Utilizando múltiplos processadores distribuídos;
- Utilizando unidades de processamento gráfico (GPGPU).

Em todas as implementações, o tamanho da população e o número de gerações são fornecidos como entrada, bem como o intervalo de migração nas implementações paralelas. A

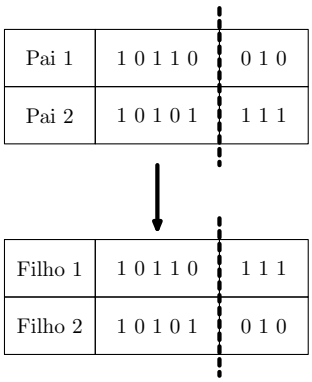

Figura 1: *Crossover* de um ponto.

representação do problema da mochila multidimensional foi feita através de uma população contendo indivíduos representados por cromossomos com genes binários. Cada cromossomo, ou indivíduo, contém n genes, onde n é o número de itens disponíveis, e representa uma possível solução para o problema.

Nas implementações realizadas, tanto a população inicial gerada aleatoriamente quanto as subsequentes são compostas apenas por indivíduos que representam uma solução viável para o problema, ou seja, uma solução válida. Para garantir que um indivíduo seja uma solução viável, logo após sua geração, verifica-se se ele satisfaz as restrições do problema e, caso não satisfaça, um item aleatório é "abandonado"; esse processo se repete até que a solução seja viável.

A seleção é feita utilizando o conceito de elitismo: os dois melhores indivíduos da população atual são escolhidos e são utilizados para dar origem a todos os indivíduos da nova geração. A estratégia de reprodução utilizada é o crossover de um ponto, ilustrado na figura 1, no qual uma posição, denominada ponto de corte, é escolhida aleatoriamente dentro dos pais e dois filhos são gerados: um contendo os primeiros genes, até a posição sorteada, provenientes de um dos pais e os restantes provenientes do segundo pai; o outro filho é obtido da maneira inversa. Para a mutação utilizou-se a estratégia invertida, na qual um bit é selecionado aleatoriamente e tem o seu valor invertido.

Todos os programas foram implementados utilizando a linguagem C++ e os detalhes de implementação de cada um dos algoritmos paralelos são abordados nas próximas subseções.

3.1. Implementação com Múltiplos Processadores

A implementação paralela para múltiplos processadores foi realizada utilizando a biblioteca de troca de mensagens MPI (*Message Passing Interface*) (Pacheco, 1997). A abordagem implementada foi a de granularidade grossa com múltiplas subpopulações, na qual cada processador executa o algoritmo sequencial. O processo de migração obedece a uma topologia de anel e, a cada intervalo de migração, os processadores trocam indivíduos com os processadores adjacentes – o seu sucessor e o antecessor. O elemento que cada processador envia para seu sucessor é aquele que possui o melhor *fitness* localmente; o elemento recebido de seu antecessor é utilizado para substituir o indivíduo local com menor *fitness*.

Os parâmetros de entrada do algoritmo são: o arquivo contendo os casos de teste, o tamanho da população, o número de gerações e o intervalo de migração. A saída gerada para cada processador mostra o tempo de execução do programa em segundos, o valor do *fitness* do melhor indivíduo e os indivíduos da última geração da população ordenada de maneira decrescente por *fitness*, ou seja, do elemento mais ao menos apto. O algoritmo 2 ilustra os passos seguidos nesta implementação.

3.2. Implementação com GPGPU

A implementação utilizando unidades de processamento gráfico (GPGPU – *General Purpose Graphics Processing Unit*) foi realizada com o auxílio da biblioteca CUDA (*Compute Unified*

Algoritmo 2: AG Paralelo de Granularidade Grossa.

```
1  Processador 0 lê dados de entrada e os distribui entre os demais processadores;
2  Cada processador gera a população inicial local;
3  Cada processador calcula o fitness dos indivíduos;
4  repita
5      Seleciona pais na população atual;
6      Reprodução;
7      Avalia o fitness da cada filho;
8      Substitui toda a população pelos novos indivíduos;
9      se iteração é múltiplo do intervalo de migração então
10         Processador 0 sorteia aleatoriamente um processador;
11         Processador sorteado envia seu melhor indivíduo para os demais;
12         Demais processadores substituem indivíduo com menor fitness pelo
           recebido;
13     fim
14 até cada processador executar o número de gerações especificado;
15 Cada processador imprime sua popuação final;
```

Device Architecture) (Nvidia, 2013), que possibilita a execução de algumas atividades utilizando os núcleos de processamento presentes em placas de vídeo da NVIDIA. A programação com CUDA permite que sejam mescladas atividades a serem executadas na CPU e atividades a serem executadas na GPGPU e, para isso, existem três modificadores que antecedem a declaração de funções: *host*, *global* e *device*. As funções *host* são executadas na CPU e somente podem ser invocadas em códigos executados na CPU. As funções *global* são executadas na GPGPU e invocadas de funções executadas na CPU, enquanto as funções *device* são executadas e invocadas na GPGPU. As funções executadas na GPGPU são denominadas de *kernel*.

Nessa abordagem, dividiu-se a população em ilhas, representadas por blocos de *threads* nos quais cada *thread* representa um indivíduo; a estratégia de migração é semelhante à da implementação em MPI, com a troca de indivíduos realizadas entre os blocos. Na etapa de geração da população inicial, cada *thread* é responsável por gerar e calcular o *fitness* de um indivíduo.

O processo de seleção é realizado por todas as *threads* simultaneamente e de maneira determinística e, então, metade das *threads* criadas realiza o *crossover* dos pais; cada *thread* gera dois indivíduos, aplica o operador de mutação e calcula o *fitness*. O controle do número de gerações é feito na CPU, responsável por gerenciar as chamadas dos *kernels* que implementam as etapas do algoritmo genético. O algoritmo 3 ilustra os passos do programa desenvolvido.

4. Resultados

Para efeitos comparativos, os programas desenvolvidos foram executados utilizando como entrada o conjunto de testes da biblioteca OR (obtido em (Drake, 2014)) por ser esse o conjunto mais utilizado nos trabalhos acerca do tema. A biblioteca é composta por instâncias de problemas com 5, 10 ou 30 recursos e 100, 250 ou 500 itens e, para cada combinação de recursos/itens também se considera, ainda, um "fator α de aperto" (*tightness factor*). O fator α determina como os valores do vetor de recursos disponíveis B se relacionam com a demanda por tais recursos, armazenados na matriz R, e pode assumir os valores 0.25, 0.50 ou 0.75. Assim, os valores em B são gerados obedecendo à equação 3.

$$b_j = \alpha * \sum_{i=1}^{n} r_{ji}, \text{para } j = 1, 2, ..., m \tag{3}$$

Por essa equação, pode-se afirmar que a instância do problema se torna mais restrita quando α tende a zero. Para cada uma das configurações $n * m * \alpha$, a biblioteca possui 10 instâncias de

Algoritmo 3: AG Paralelo de Granularidade Fina com CUDA.

```
1  Leitura dos dados de entrada na CPU;
2  Criação de n threads na GPGPU para gerar um indivíduo da população inicial;
3  Cada thread calcula o fitness do indivíduo por ela gerado;
4  repita
5      Todas as threads selecionam os pais na população atual;
6      Metade das threads realiza a reprodução, cada uma gera dois novos indivíduos;
7      Cada thread avalia o fitness de seus indivíduos;
8      Toda a população é substituída pelos novos indivíduos;
9  até o número de gerações especificado seja alcançado por todas as threads;
10 População final é copiada para CPU;
11 CPU imprime população final;
```

teste, totalizando 270 diferentes problemas. O programa sequencial e o paralelo com MPI foram executados considerando uma população de tamanho igual a 4096 e 50 gerações; o programa em MPI considera também um intervalo de migração de 10 gerações. Para o programa em MPI, foram realizadas execuções com 2, 8, 32 e 64 processadores e o ambiente utilizado para execução consiste em um *cluster beowulf* contendo:

- 64 nós de processamento com 4 x 2.2 GHz Opteron Cores com 8 GB de RAM por nó;
- 4 nós com 12 x 2.53 GHz Intel Xeon E5649 com 24 GB de RAM por nó;
- 3 TB de armazenamento em disco;
- Interconexão usando gigabit ethernet.

O *cluster* em questão utiliza um gerenciador de submissão de processos, o que torna transparente o modo como foi feita a alocação dos processadores para os processos.

O programa em CUDA foi executado utilizando uma distribuição contendo 8 blocos de 512 *threads* cada, totalizando 4096 indivíduos na população; o programa foi executado por 50 gerações. O ambiente de execução utilizado para o programa sequencial e para o que utiliza CUDA é caracterizado por:

- Processador Intel I7-3770 de 3.40 GHz;
- 24 GB de RAM;
- 256 GB de armazenamento SSD;
- Placa de vídeo NVIDIA GeForce GT 640 com 1 GB de memória e 384 núcleos de processamento.

Os gráficos apresentados nas figuras de 2 a 4 ilustram as médias dos tempos obtidos pela execução de cada programa para cada uma das instâncias, agrupadas por número de recursos; no topo de cada um dos gráficos estão os tempos de execução do algoritmo sequencial e do algoritmo utilizando CUDA. Para avaliar a qualidade das soluções obtidas, foi calculada, para cada instância, a porcentagem da diferença entre a melhor solução conhecida e a obtida, essa diferença é chamada de *gap*. As figuras de 5 a 7 contêm os gráficos que ilustram os *gaps* médios das soluções obtidas agrupados por número de recursos.

Pela análise dos gráficos, pode-se notar que a utilização do paralelismo é viável, causando, entretanto, redução na qualidade das soluções obtidas. De outro lado, pode-se ver também que o

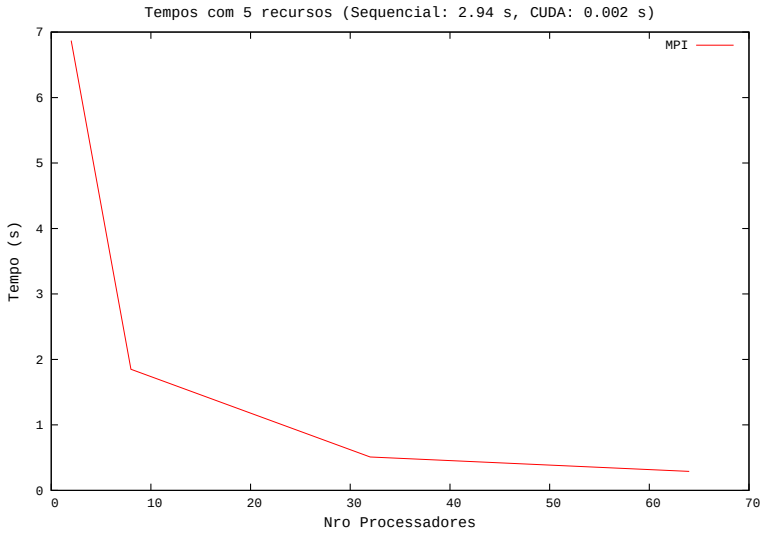

Figura 2: Tempos com as instâncias com 5 recursos.

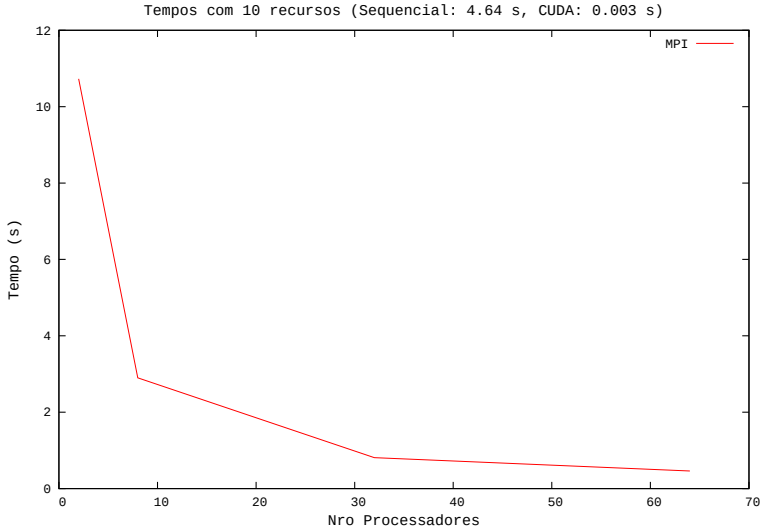

Figura 3: Tempos com as instâncias com 10 recursos.

tempo de execução reduz-se à medida que o número de processadores aumenta, e em uma escala maior do que a redução da qualidade; essa característica pode justificar o uso do paralelismo quando o que importa é a obtenção de uma solução no menor tempo possível, mesmo que com pequenas perdas. Como uma forma de avaliar as razões para a perda de qualidade nas versões paralelas, serão realizados novos testes com diferentes configurações de população, de distribuição dos dados e de intervalos de migração, bem como fazendo uso de outras técnicas para realizar tal migração.

5. Considerações Finais

A utilização de algoritmos genéticos na solução de problemas, para os quais a obtenção de uma solução exata possui um custo computacional elevado, tem obtido resultados importantes e vem sendo bastante estudada nos últimos anos. Uma classe de problemas que tem explorado bastante os algoritmos genéticos é a otimização combinatória. Com a grande disponibilidade de recursos computacionais de alto desempenho há o desafio de explorar o paralelismo dessa heurística. O presente trabalho descreve estratégias de paralelização em dois modelos computacionais paralelos para a heurística de algoritmos genéticos na solução do problema da mochila multidimensional.

Para comparar os resultados obtidos utilizou-se uma versão sequencial de um algoritmo

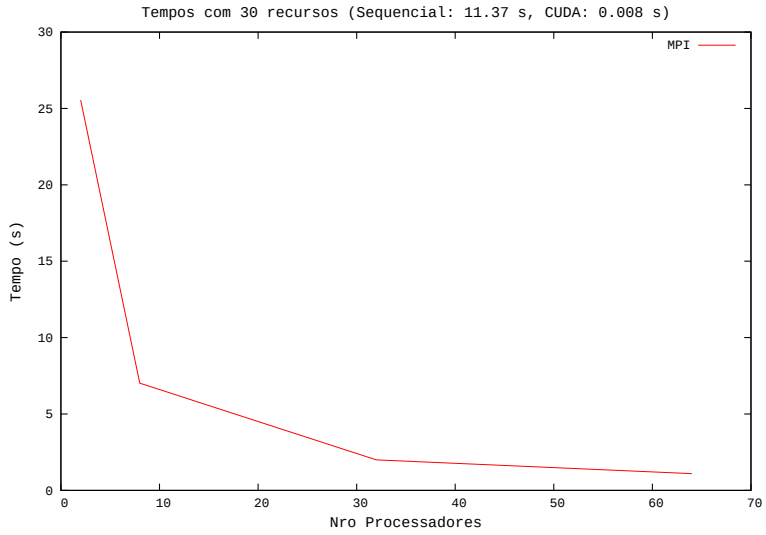

Figura 4: Tempos com as instâncias com 30 recursos.

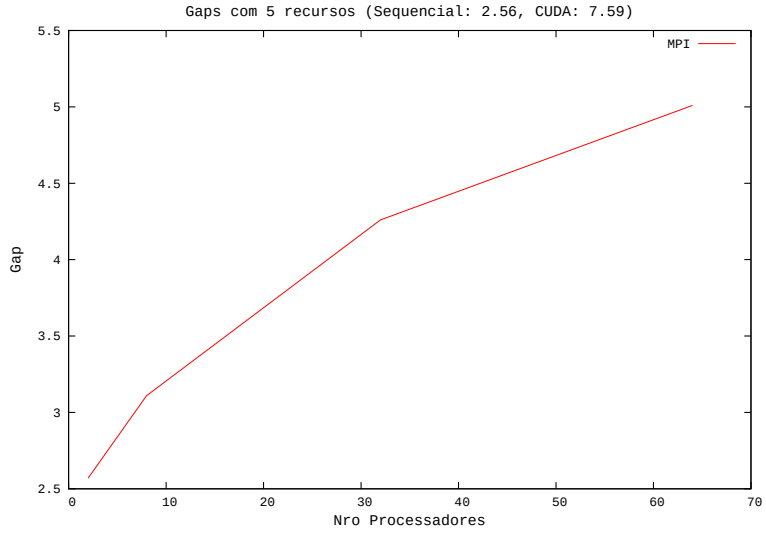

Figura 5: *Gaps* com as instâncias com 5 recursos.

genético bem simples para a solução do problema da mochila multidimensional. Foram implementadas duas versões paralelas, uma no modelo de memória distribuída e a outra no modelo de memória compartilhada. Todos os programas foram escritos utilizando a linguagem C++ e as implementações paralelas utilizaram as bibliotecas MPI e CUDA.

Os gráficos obtidos com as execuções dos programas no modelo de memória distribuída indicam que a utilização de estratégias de paralelização torna os tempos de execução menores quando utilizamos maior número de processadores, entretanto, não foi possível observar vantagem na associação de *threads* na execução local das tarefas. Em função do baixo *speedup*, os resultados relacionados às implementações com MPI e *threads* não estão descritos nesse trabalho. A implementação no modelo de memória compartilhada usou GPGPU e a biblioteca CUDA, e obteve resultados relativos à qualidade das soluções semelhantes às da implementação em memória distribuída; no entanto, obteve tempos de execução bem melhores. Nota-se também um grande potencial de obtenção de versões do algoritmo com desempenho superior, tanto em qualidade da solução como em tempo de execução, aprimorando a política de utilização e acesso às hierarquias de memórias disponíveis. As mesmas técnicas de paralelização utilizadas neste trabalho podem ser

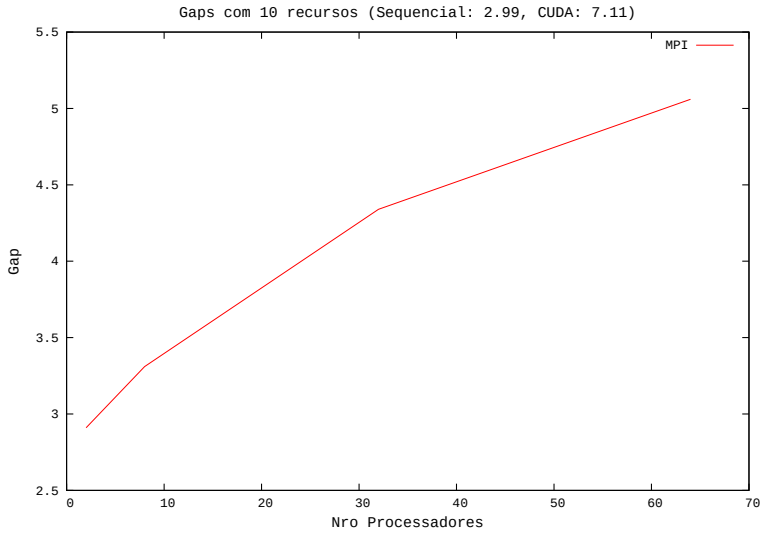

Figura 6: *Gaps* com as instâncias com 10 recursos.

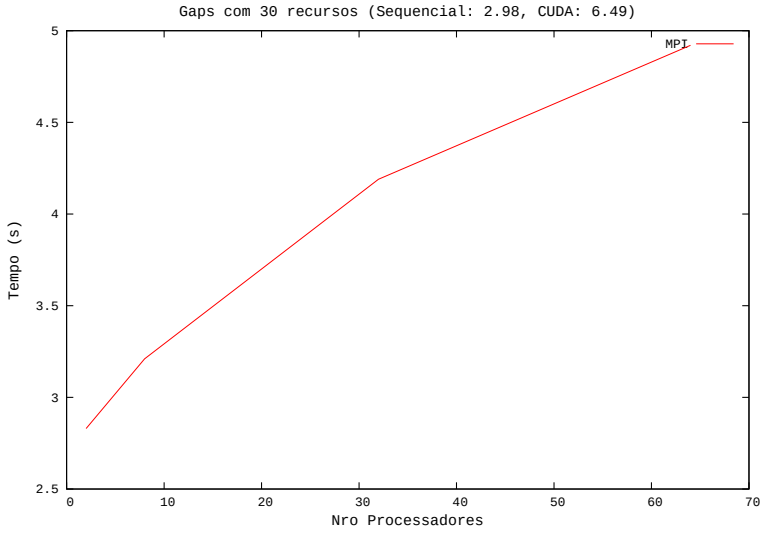

Figura 7: *Gaps* com as instâncias com 30 recursos.

estendidas para algoritmos genéticos mais complexos e com isso explorar melhores resultados para o problema da mochila multidimensional. Os resultados obtidos neste trabalho indicam um grande potencial na paralelização de algoritmos genéticos.

Como trabalhos futuros, a exploração de diferentes alternativas para realizar os processos de seleção, reprodução, mutação e migração do algoritmo genético, bem como revisão dos códigos dos programas para verificar se algumas tarefas podem ser realizadas de maneira mais eficiente são possibilidades que serão estudadas. Outra alternativa a ser explorada é testar diferentes formas de mapeamento das *threads* no programa em CUDA e de utilização da hierarquia de memória.

Referências

Barney, B., *POSIX Threads Programming*, disponível em: computing.llnl.gov/tutorials/pthreads/, 2013.

Beasley, J., *OR-Library*, Disponível em: people.brunel.ac.uk/˜mastjjb/jeb/info.html, 2013.

Burns, A., *A Parallel Algorithm for the Multidimensional Knapsack Problem*, Master Thesis, East Stroudsburg University, 1993.

Cáceres, E. e Nishibe, C., 0-1 Knapsack Problem: BSP/CGM Algorithm and Implementation, *Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2005)*, 331-335, 2005.

Cantu-Paz, E., A survey of parallel genetic algorithms. *Calculateurs Paralleles, eseaux et Systems Repartis*, v. 10, n. 2, 1998.

Chu, P. e Beasley, J., A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, v. 4, 1998.

Drake, J., *MKP Instances*. Disponível em: http://www.cs.nott.ac.uk/ jqd/mkp/index.html, 2014.

Fingler, H. *Otimização de Colônias de Formigas em CUDA: O Problema da Mochila Multidimensional e o Problema Quadrática de Alocação*, Dissertação de Mestrado, Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, 2013.

Haupt, R. e Haupt, S., *Practical Genetic Algorithms*. 2. ed., Wiley-Interscience, 2004.

Holland, J., *Adaption in Natural and Artificial Systems*, The University of Michigan Press, 1975.

Lebedev, S. S., Combinatorial Methods of Discrete Programming, *Proceedings of 1968 Alma-Ata Summer School on Mathematical Programming – U.S.S.R. Academy of Sciences*, 38-88, 1969.

Liu, R. T. e Lv, X. J., MapReduce-Based Ant Colony Optimization Algorithm for Multi-Dimensional Knapsack Problem, *Applied Mechanics and Materials*, Vols. 380-384, 1877-1880, 2013.

Niar, S. e Freville, A., A Parallel Tabu Search Algorithm for The 0-1 Multidimensional Knapsack Problem. IPPS' 97 *Proceedings of the 11th International Symposium on Parallel Processing*, 512-516, 1997.

Nvidia. *Developer Zone*, disponível em: https://developer.nvidia.com/category-/zone/cuda-zone, 2013

Pacheco, P., *Parallel Programming with MPI*, Morgan Kaufmann Publishers, 1997.

Posadas, C. B., Ayala, D. V., Garcia, J. S., Silverio, S. L. e Vidal, M. T., A Solution to Multidimensional Knapsack Problem Using a Parallel Genetic Algorithm, *International Journal of Intelligent Information Processing*, Volume 1, Number 2, 47-54, 2010.

Solnon, C. e Bridge, D., An Ant Colony Optimization Meta-heuristic for Subset Selection Problems, *Systems Engineering Using Swarm Particle Optimization*, N. Nedjah and L. M. Mourelle, Eds., Nova Science Publishers, 7-29, 2006.