UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS RIO PARANAÍBA

GUILHERME MARTINS 2762
GUILHERME HENRIQUE ALVES DE PAIVA 2791

ABORDAGEM HEURÍSTICA PARA O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS EM CENTROS DE *CROSS-DOCKING* COM MÚLTIPLAS DOCAS UTILIZANDO PROGRAMAÇÃO PARALELA EM GPU

RIO PARANAÍBA
2016

GUILHERME MARTINS 2762
GUILHERME HENRIQUE ALVES DE PAIVA 2791

ABORDAGEM HEURÍSTICA PARA O PROBLEMA DE SEQUENCIAMENTO DE TAREFAS EM CENTROS DE *CROSS-DOCKING* COM MÚLTIPLAS DOCAS UTILIZANDO PROGRAMAÇÃO PARALELA EM GPU

Monografia, apresentada a Universidade Federal de Viçosa, como requisito para obtenção do título de bacharel em Sistemas de Informação

Orientador: Guilherme de Castro Pena

RIO PARANAÍBA

2016

Resumo

O presente trabalho tem como objetivo apresentar uma solução heurística, baseada na metodologia *multi-start*, para o problema de *flow shop* com restrições de *Cross-docking* com múltiplas docas. *Cross-docking* é formulado como um problema de sequenciamento de fluxo de tarefas em um centro de distribuição, que possui restrições na configuração das máquinas de entrada e saída. O cenário abordado neste trabalho trata-se de um centro de distribuição que possui múltiplas máquinas de entrada e múltiplas máquinas de saída. A metodologia utilizada na resolução do problema em questão é fundamentada na técnica *multi-start*, onde duas heurísticas construtivas, LPT (*Longest Processing Time* - Maior Tempo de Processamento) e LNS (*Largest Numbers of Sucessors First* - Maior Número de Sucessores Primeiro), geram soluções viáveis iniciais que são refinadas por meio das técnicas de busca local *Swap* e *Shift*. Para validação da abordagem, implementou-se um *lowerbound* descrito na literatura. Os algoritmos foram desenvolvidos usando uma abordagem sequencial em CPU (*Central Processing Unit* - Unidade de Processamento Central) e uma abordagem paralela em GPU (*Graphics Processing Unit* - Unidade de Processamento Gráfico), através da ferramenta CUDA (*Compute Unified Device Architecture* - Arquitetura de Dispositivos Computacionais Unificada). As soluções finais tiveram GAPs (*Gain Average Percentage* - Porcentagem Média de Ganho) significativos em relação ao *lowerbound* gerado e verificou-se ganhos (*SpeedUPs*) de até 2x.

Palavras-chaves: sequenciamento, *cross-docking*, máquinas paralelas, computação paralela, GPGPU, CUDA.

Abstract

This document aims to present a heuristic solution, based on multi-start methodology for the problem of flow shop with Cross-docking constraints with multiple docks. Cross-docking is formulated as a job flow sequencing problem in a distribution center, which has constraints on the configuration of input and output machines. The problem addressed in this paper describes a distribution center, which has several input machines and several output machines. The methodology used for solving the problem is based on the multi-start technique, where two constructive heuristics, LPT (Longest Processing Time) and LNS (Largest Numbers of Sucessors First), generate initial feasible solutions that are refined by local search techniques, Swap and Shift. To validate the approach, a lowerbound described in literature was implemented. The algorithms were developed using a sequential approach in CPU (Central Processing Unit) and a parallel approach in GPU, through the CUDA (Compute Unified Device Architecture) tool. The final solutions had significant GAPs (Gain Average Percentage) and SpeedUPs gains of up to 2x were verified.

Key-words: scheduling, flow shop, cross-docking, parallel machines, parallel computing, GPGPU, CUDA.

Lista de ilustrações

Lista de tabelas

Lista de siglas

GPU	*Graphics Processing Unit* - Unidade de Processamento Gráficos
CUDA	*Compute Unified Device Architecture* - Arquitetura de Dispositivos Computacionais Unificada
GAP	*Gain Average Percentage* - Média Percentual de Ganho
CCD	Centro de *Cross-docking*
CD	Centro de *Cross-docking*
GPGPU	*General Purpose on Graphics Processing Unit* - Unidades de Processamento Gráfico para Propósito Geral
CPU	Central Processing Unit - Unidade Central de Processamento
API	*Application Programming Interface* - Interface de Programação de Aplicativos
LPT	*Longest Processing Time - Maior Tempo de Processamento*
LNS	*Largest Numbers of Successors First* - Maior Número de Sucessores Primeiro

Lista de algoritmos

Sumário

1 Introdução

O presente trabalho aborda uma variação do problema de Sequenciamento de Tarefas (*Scheduling*), dadas as especificações de *Flow shop* e *Cross-docking*. *Flow shop* trata-se de uma técnica de alocação de recursos em uma sequência correta de forma a atingir um objetivo operacional específico (VALLADA; RUIZ; FRAMINAN, 2015). *Cross-docking* pode ser definido como uma metodologia que define um conjunto de procedimentos logísticos que visam a redução de custos operacionais e aumento dos lucros através da otimização dos processos relacionados a armazenagem e transporte de produtos (KINNEAR, 1997). Sendo assim, o problema abordado trata-se do sequenciamento de recursos (*jobs*) em um centro de *Cross-docking* com múltiplas docas (máquinas) de entrada e múltiplas docas (máquinas) de saída.

Em função dos recentes e relevantes resultados disponíveis na literatura acerca da utilização do paradigma de computação paralela em Unidade de Processamento Gráfico (*Graphics Processing Unit* - GPU), através do modelo CUDA, na resolução de problemas de Pesquisa Operacional (BOYER; BAZ, 2013), visa-se construir uma solução para resolução de uma variante do problema de *Cross-docking*, utilizando os recursos do referido modelo.

O problema abordado trata-se de um problema NP-Difícil (CHEN; SONG, 2009), não se conhecendo ainda uma abordagem que o solucione de forma exata, apresentando sua solução ótima, em tempo polinomial. Sendo assim, tem-se como objetivo propor uma solução heurística, uma técnica capaz de oferecer respostas viáveis, para a variante do problema abordado, através da implementação de um algoritmo, baseado no modelo *multi-start*, que avalia múltiplas soluções iniciais para um cenário, estruturado segundo o paradigma de programação paralela em GPU, utilizando os recursos da ferramenta CUDA.

Foi validada, então, a solução apresentada, comparando seu desempenho computacional com soluções apresentadas na literatura.

2 Objetivos

O objetivo principal deste trabalho é propor uma heurística para resolução de uma especificação do problema de *Flow Shop* com restrição de *Cross-docking* (CCD), com várias docas de entrada e saída, através do desenvolvimento de um algoritmo baseado em computação paralela em GPU.

2.1 Objetivos Específicos

- Realizar um estudo sobre o funcionamento e especificidades dos processos do tipo *Flow Shop*;
- Realizar um estudo sobre as particularidades e características do *Cross-docking*, no intuito de estabelecer a compreensão do tema;
- Fazer uma revisão na literatura relativa às áreas de interesse e avaliar as melhores soluções existentes até o momento;
- Realizar um estudo sobre o funcionamento das ferramentas a serem utilizadas no desenvolvimento do trabalho, sendo estas, CUDA e GPU e, desta forma, avaliar e compreender as melhores e mais utilizadas soluções no que se refere à programação paralela em GPUs;
- Implementar uma solução heurística para o problema especificado;
- Avaliar a qualidade dos resultados obtidos comparando-os com os existentes na literatura;
- Desenvolver e submeter um artigo contendo os resultados obtidos em conferência do tema abordado.

3 Referencial Teórico

3.1 Sequenciamento(*Scheduling*)

Segundo Pinedo (2008), sequenciamento pode ser definido como o processo de tomada de decisão, amplamente praticado na indústria e mercado, que envolve correta alocação de recursos para minimização dos custos ao se realizar um conjunto de tarefas.

No ambiente competitivo dos dias atuais, instituições de diferentes categorias utilizam técnicas de sequenciamento para reduzir custos e maximizar seus lucros operacionais (BELLE; VALCKENAERS; CATTRYSSE, 2012).

Existem diversas configurações possíveis para que todas as tarefas de um processo sejam executadas, entretanto, existe uma ou mais formas (sequências) que garantem o melhor resultado possível. Este é o objetivo geral visado pela sistematização do sequenciamento (PINEDO, 2008).

O método gráfico desenvolvido por Henry Gantt é um dos principais recursos utilizados para ilustrar e analisar os problemas de sequenciamento.

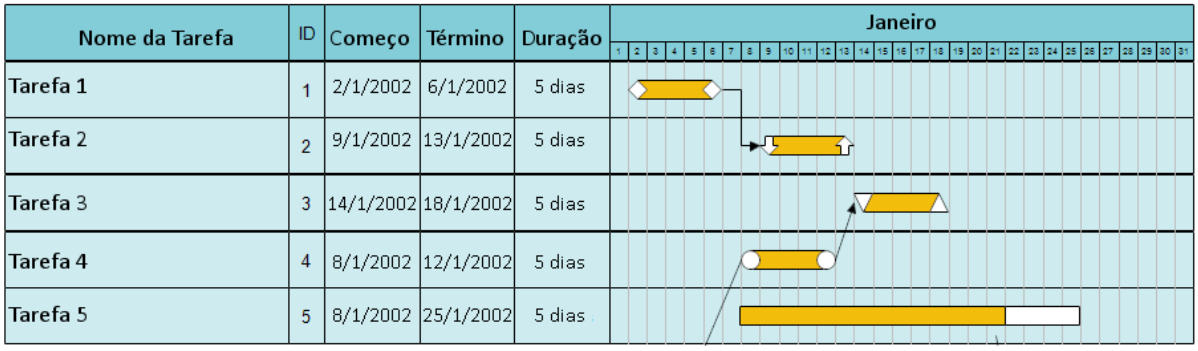

Figura 1 – Gráfico de *Gantt*

Adaptado do endereço eletrônico: <http://www.rff.com/gantt1.htm>

No gráfico de Gantt, apresentado através da Figura 1, é possível identificar 5 tarefas que possuem um tempo de início e fim. Cada tarefa pode ou não depender de outra, o que é representado por meio de setas. A cor amarela representa a quantidade de cada atividade que foi executada, e a cor branca representa o quanto ainda está pendente.

3.2 *Job Shop*

Kundakci e Kulak (2016) definem *job shop* como uma derivação do problema de sequenciamento, o qual está entre os problemas mais desafiadores de otimização. Este problema envolve determinar um sequenciamento para *jobs* (itens, tarefas...) que possuem uma sequência de operações pré-especificadas em um ambiente com multi-máquinas. Ele se refere à um problema estático onde um conjunto de *jobs* são processados em uma série de máquinas. Seu principal objetivo é encontrar uma sequência que processa todos os *jobs* de maneira que otimiza o objetivos de desempenho.

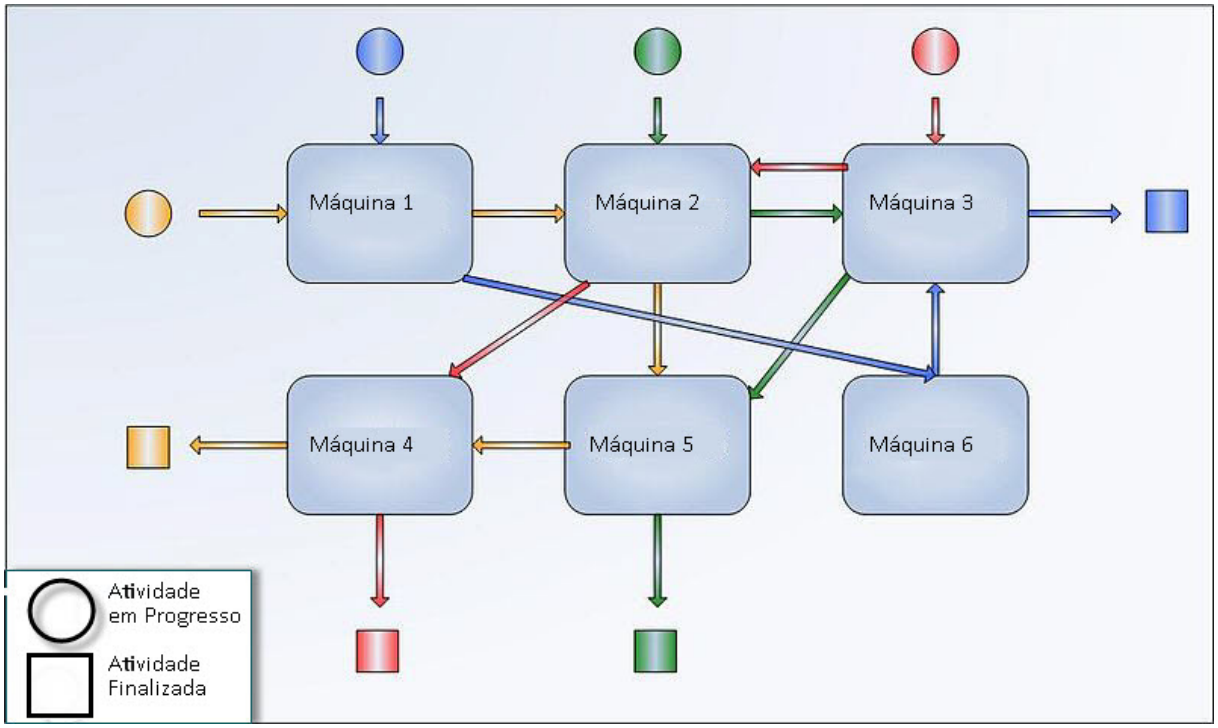

Figura 2 – Diagrama de *Job Shop*

Adaptado do endereço eletrônico: <http://www.bizharmony.com/Blog/September-2013/monetize-your-abandoned-or-dormant-technologies.aspx>

Como mostrado no diagrama da Figura 2, *job shops* podem trabalhar com múltiplos processos em múltiplos projetos simultaneamente.

3.3 *Flow Shop*

Vallada, Ruiz e Framinan (2015) definem o problema de *flow shop* como uma metodologia que visa determinar uma sequência de processamento de um grupo de n *jobs* (tarefas) em um conjunto de m máquinas no intuito de minimizar o tempo ocioso e o tempo de espera do processamento completo, desta forma, busca-se reduzir o tempo

gasto desde o início da execução do primeiro *job* na primeira máquina e o fim da execução do último *job* na última máquina, definido como *makespan*.

Cada job possui tempo diferente de processamento em cada uma das máquinas, o número de *jobs* executado em uma máquina por vez deve ser somente 1 e, além disso, cada *job* é executado sequencialmente, iniciando seu processamento desde a primeira máquina até a última, ou seja, um *job* não pode iniciar seu processamento em uma máquina posterior sem antes concluir sua execução na máquina anterior, conforme ilustrado na figura 3.

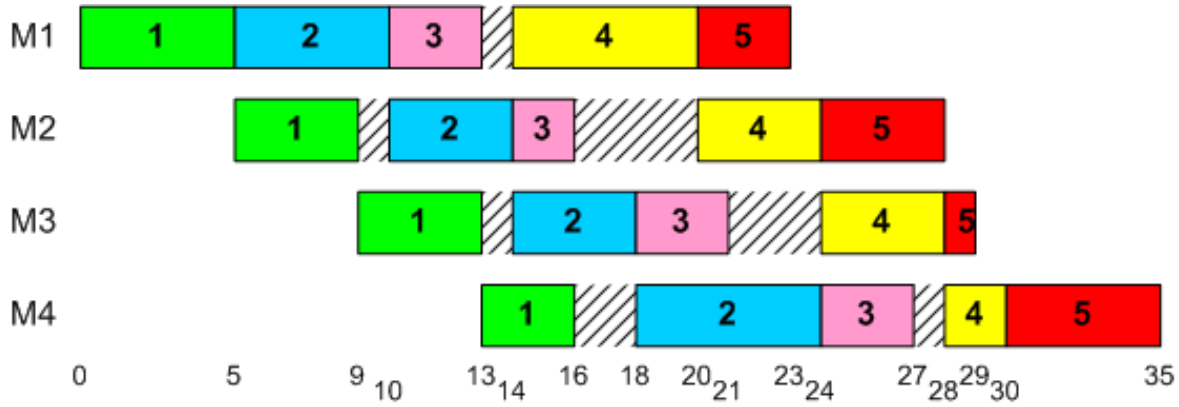

Figura 3 – Diagrama de *Flow Shop*

Fonte: <http://www.bizharmony.com/Blog/September-2013/monetize-your-abandoned-or-dormant-technologies.aspx>

3.4 *Cross-docking*

Cross-docking ou *Crossdocking* pode ser definido, de uma forma objetiva, como o processo de receber produtos ou matéria-prima de vários fornecedores diferentes, reuní-los e enviar para um destino comum, evitando ou reduzindo o processo de armazenagem. (KINNEAR, 1997).

Cross-docking é um conceito logístico que trata da integração de nós intermediários em uma rede de transporte. Em um terminal ou Centro de *Cross-docking*, carregamentos entregues por caminhões de entrada são coletados, ordenados por destino, transportados através do terminal para serem diretamente carregados em caminhões de saída, que, imediatamente, dirigem-se aos consumidores-destino. Em contraste com os métodos tradicionais de armazenagem, a estocagem dos produtos é reduzida ao máximo possível e os mesmos, usualmente, deixam o Centro de *Cross-docking* em no máximo 24 horas (STEPHAN; BOYSEN, 2011). A Figura 4 ilustra o processo de *Cross-docking*:

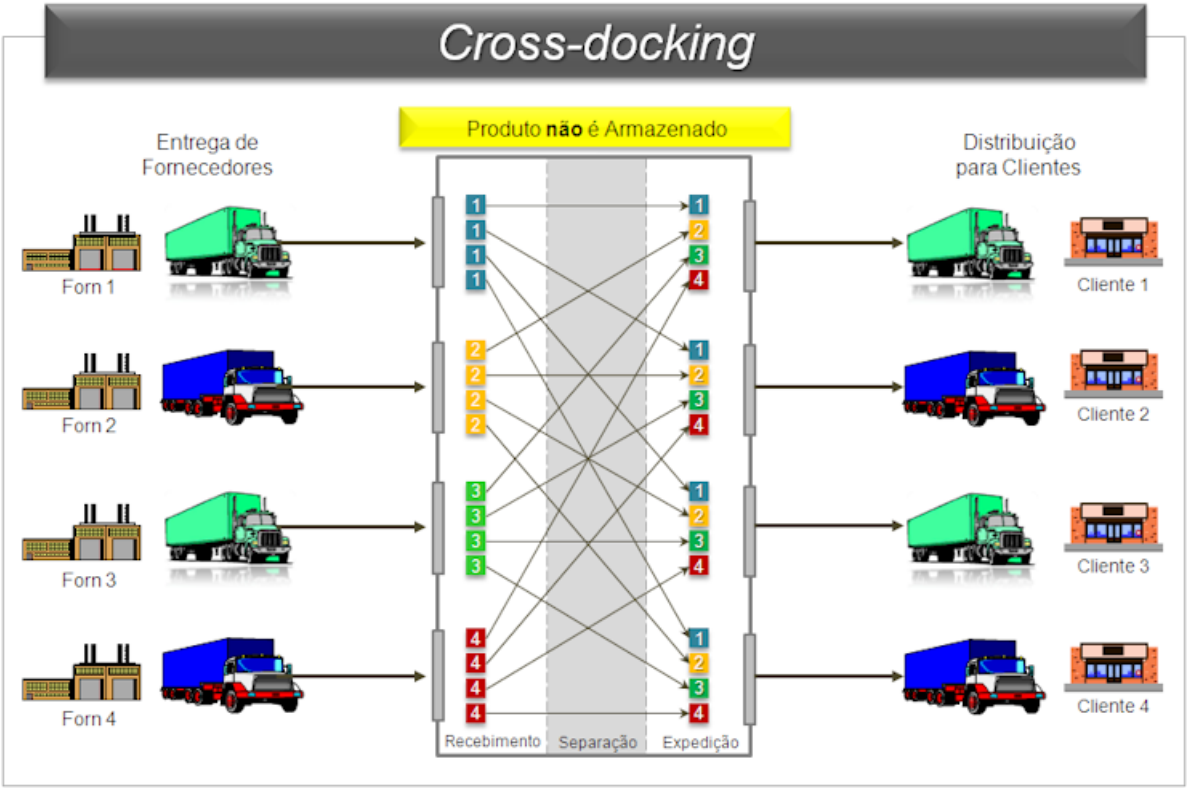

Figura 4 – Diagrama da operação de *Cross-docking*

Fonte: <http://portallogistico.com.br/2015/04/27/cross-docking-39781/>

A utilização do *Cross-docking* tem pré-requisitos e consequências favoráveis e desfavoráveis. Os principais benefícios estão relacionados à economia com custos de armazenagem. Entretanto, existem desvantagens relativas à complexidade de implementação e custos extras com transporte. Nogueira (2012) elabora uma compilação mais extensa sobre benefícios e desvantagens do referido procedimento, destacados abaixo.

Vantagens do *Cross-docking*:

- Aumenta velocidade do fluxo de produtos e circulação do estoque;
- Reduz custo de estoque;
- Dá suporte às estratégias de *just-in-time* [1];
- Promove melhor utilização dos recursos;
- Reduz necessidade de espaço;
- Reduz danos aos produtos por causa do menor manuseio;

[1] sistema de administração da produção que determina que tudo deve ser produzido, transportado ou comprado na hora exata

- Reduz furtos e compressão dos produtos;
- Reduz obsolescência (e problemas com prazo de validade) dos produtos;
- Acelera pagamento ao fornecedor, logo, melhora parcerias;

Desvantagens do *Crossdocking*:

- Dificuldade de determinação dos produtos candidatos;
- Requer sincronização dos fornecedores e demanda;
- Relações imperfeitas com fornecedores;
- Pequena ou nenhuma credibilidade nos fornecedores;
- Relutância dos fornecedores quanto à eficiência do processo;
- Sindicatos temem perda de empregos;
- Dependências inadequadas ou retornos sobre investimentos insuficientes para justificar a compra, reforma ou construção de um CD (Centro de *Cross-docking*) apropriado;
- Sistemas de informação inadequados;
- Gerência nem sempre possui uma visão holística e orientada da cadeia de suprimentos;
- Medo de *stock-out* [2] pela ausência de estoque de segurança.

Consequente à avaliação das vantagens e desvantagens da implantação do *Cross-docking*, faz-se necessário analisar quando utilizar o referido sistema. Apte e Viswanathan (2000) propõem a análise da adequação do *Cross-docking* baseado em função do custo de *stock-out* e a taxa de demanda por produto. Desta forma, apenas o equilíbrio entre os dois fatores favorece a implantação do *Cross-docking*, conforme pode ser verificado na figura 5.

[2] consequências econômicas por não ser capaz de satisfazer uma demanda interna ou externa por um produto ou serviço

Custo de stock-out	Taxa de demanda por produto: Estável ou constante	Taxa de demanda por produto: Instável ou variável
Alto	Cross-docking pode ser Implementado	Favorável à implantação de métodos tradicionais
Baixo	Favorável à implantação do Cross-docking	Cross-docking pode ser implementado

Figura 5 – Quando utilizar o *Cross-docking*

Adaptado de Belle, Valckenaers e Cattrysse (2012)

Em vista das vantagens logísticas apresentadas anteriormente, *Cross-docking* pode ser uma estratégia interessante para empresas, propiciando importantes vantagens competitivas. São exemplos de companhias que implementaram com sucesso a referida solução *Wal Mart* (KINNEAR, 1997) e *Goodyear GB Ltd.* (STALK; EVANS; SHULMAN, 1991).

Ao longo dos anos, diversos algoritmos foram propostos para solução ótima ou heurística do problema do *Cross-docking*, dada suas características e restrições, fato que torna complexo o referido problema e demanda soluções específicas para cada uma de suas variantes. Dentre as mais recentes abordagens presentes na literatura, podemos citar a heurística híbrida, utilizando arrefecimento simulado (*Simulated Annealing*) e busca-tabu (*Tabu Search*), proposta por Küçükoğlu e Öztürk (2015), para solução do problema de roteamento de veículos e embalagem de produtos; as heurísticas propostas por Liao, Egbelu e Chang (2013) para sequenciamento de veículos de entrada em um centro de *Cross-docking* com múltiplas docas; o algoritmo genético adaptativo para sequenciamento de veículos em ambiente similar ao citado anteriormente proposto por Kim e Joo (2015) e a heurística proposta por Shakeri et al. (2012) para solucionar o sequenciamento de veículos em Centros de *Cross-docking* com limitação de recursos. Belle, Valckenaers e Cattrysse (2012) apresentam uma listagem extensa das soluções existentes na literatura, catalogadas de acordo com as restrições e características do problema, além do modelo de algoritmo proposto.

Chen e Song (2009) apresentam a modelagem para o problema de sequenciamento de máquinas em *cross-docking* com múltiplas docas, incluindo os cálculos de *lowerbound* e GAP, que serão posteriormente utilizados para validação do presente trabalho. Este trabalho e outros citados na seção atual serão discutidos com mais intensidade na Seção 4.

3.5 Programação Paralela

Almasi e Gottlieb (1989) definem computação paralela como o tipo de computação onde vários cálculos e operações são realizadas simultaneamente, partindo do princípio de que problemas complexos podem ser subdivididos em problemas menores. Os problemas fracionados são processados paralelamente em vários núcleos de processamento ou computadores e, no fim, tem-se a reunião do resultado do processamento das partes obtendo-se o resultado final.

A programação paralela pode ser definida como o paradigma ou modelo de abstração baseado nos conceitos de computação paralela, onde o processamento de um código fonte é dividido entre vários processadores ou computadores (KUMAR et al., 1994).

Tradicionalmente, os softwares são construídos em computação serial ou sequencial, onde o problema é divido em um série discreta de instruções, que são executadas sequencialmente, uma após a outra, em um único processador, onde somente uma instrução pode ser executada em um dado intervalo de tempo ou ciclo de *clock*. Na computação paralela, existe o uso simultâneo de múltiplos recursos computacionais no intuito de resolver determinado problema. Desta forma, os problemas são divididos em partes menores que podem ser resolvidas concorrentemente, cada parte é subdividida em uma série de instruções, que são executadas simultaneamente em diferentes processadores e coordenadas por um mecanismo de controle global (BARNEY, 2010).

A Figura 6 ilustra o processamento paralelo do problema de realizar o pagamento de n funcionários (f) de uma empresa, onde a função fazer pagamento é subdivida em instruções (t) que são executadas simultaneamente em diferentes processadores.

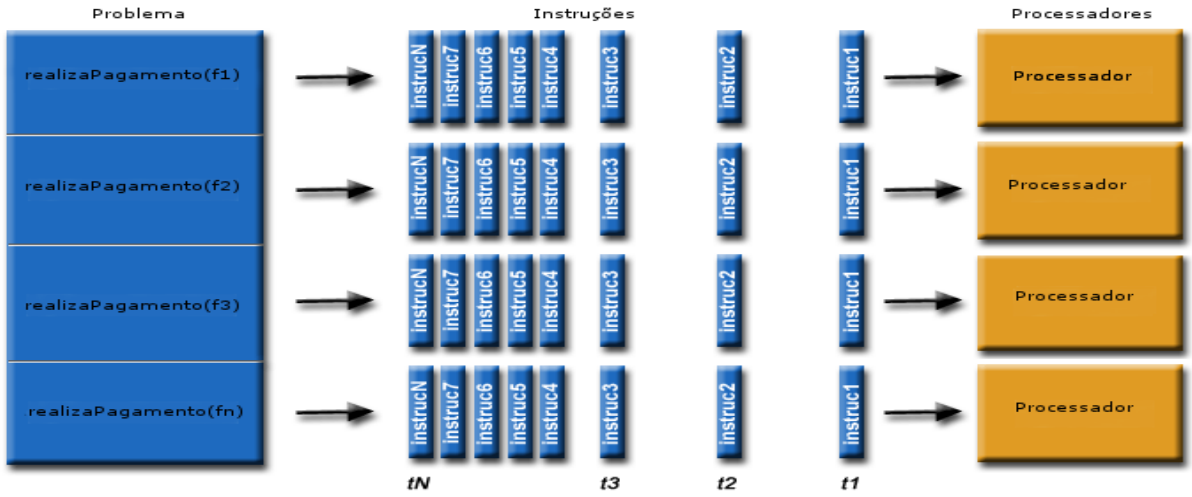

Figura 6 – Processamento paralelo de um problema

Adaptado do endereço eletrônico:
<https://computing.llnl.gov/tutorials/parallel_comp/>

O objetivo do uso de algoritmos paralelizados para solução de problemas é, segundo Kumar et al. (1994), proporcionar gasto inferior de tempo, quando comparados com soluções sequenciais, e permitir a resolução de empecilhos excessivamente complexos, cuja resposta não seja contemplada pelas metodologias tradicionais de computação, que se restringem ao processamento sequencial.

3.6 GPGPU

GPGPU (General Purpose on Graphics Processing Unit) ou Unidades de Processamento Gráfico para Propósito Geral, trata-se do termo que representa o desenvolvimento de abordagens paralelas para vários tipos de problema por meio do uso de dispositivos de processamento gráfico, os quais estão presentes na maioria dos computadores, atualmente. O processamento de grandes volumes de dados combinando-se abordagens híbridas (sequenciais e paralelas) tem sido amplamente utilizado para objetivos científicos. (MITTAL; VETTER, 2015).

As GPUs mais recentes possuem milhares de núcleos de processamento, o que as torna dispositivos especializados em processamento massivamente paralelo. Entretanto, alguns tipos de instruções que exigem desvio de fluxo na sua execução são melhor tratadas por unidades central de processamento (CPU - *Central Processing Units*), o que requer um modelo de abstração que preza pela sincronização da operação paralela em diferentes dispositivos. Desta forma, uma aplicação deve ter o processamento de suas instruções divididas entre a CPU e a GPU, de forma a otimizar o aproveitamento dos recursos computacionais característicos de ambos equipamentos. (BOURGOIN; CHAILLOUX; LAMOTTE, 2014)

O processo de GPGPU é ilustrado na Figura 7, onde o código-fonte de uma aplicação é dividido em duas partes principais: códigos obrigatoriamente sequenciais (que são melhor processados pela CPU) e funções de computação intensiva (que podem ser processadas paralelamente pela GPU). Esta aplicação é, então, processada simultaneamente utilizando os recursos da CPU e GPU.

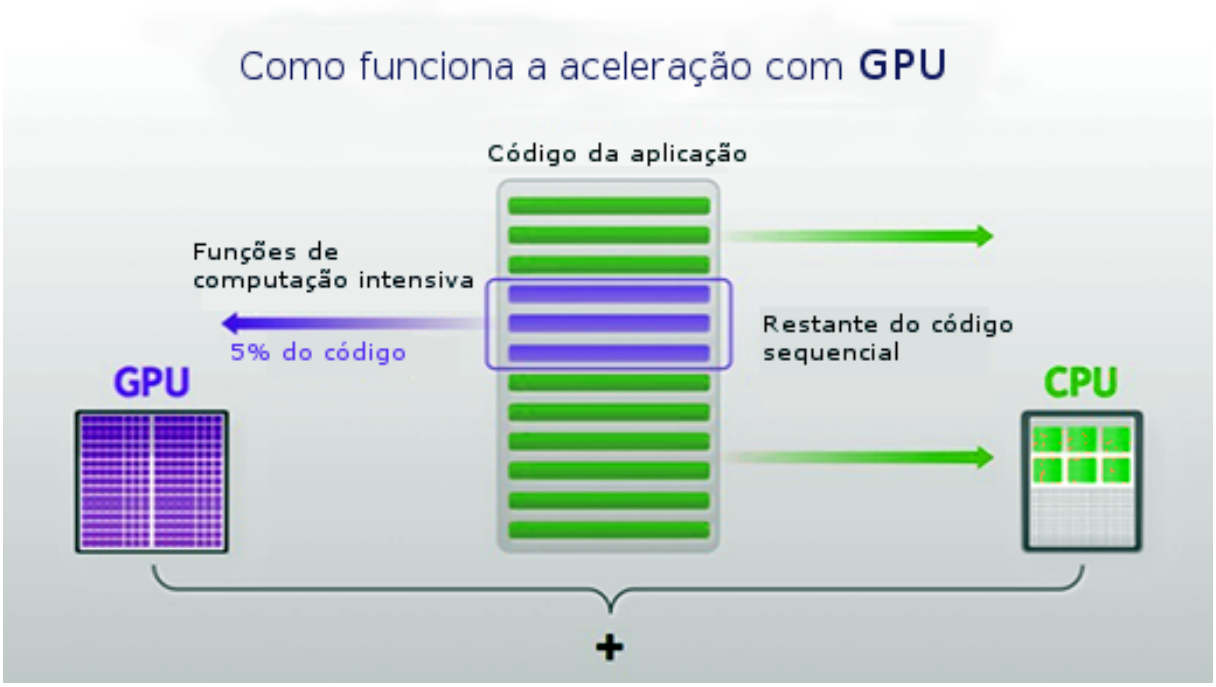

Figura 7 – Como funciona a GPGPU

Adaptado do endereço eletrônico:
<http://www.nvidia.com/object/what-is-gpu-computing.html>

Atualmente os modelos de programação paralela em GPU mais utilizados são o OPEN CL e o CUDA (BOYER; BAZ, 2013).

3.7 CUDA

Compute Unified Device Architeture ou Arquitetura de Dispositivos Computacionais Unificada (CUDA) é um conceito apresentado pela fabricante de dispositivos NVIDIA® no ano de 2006.

Trata-se de um modelo ou arquitetura de computação paralela em dispositivos de processamento gráfico (GPUs) proprietário, que oferece um conjunto de bibliotecas, APIs (*Application Programming Interface* - Interface de Programação de Aplicativos) e ferramentas que permitem a implementação de algoritmos paralelizados em GPU, utilizando as linguagens de programação C, C++ ou FORTRAN. (NVIDIA, 2015)

Os códigos em linguagem C, dispostos abaixo, ilustram os procedimentos de incrementar um vetor (array) de elementos utilizando CPU (Algoritmo 1) e programação paralela em CUDA (Algoritmo 2), onde podemos notar que no Algoritmo 2, não existe o uso da estrutura de repetição, que percorre cada posição do vetor como ilustrado no Algoritmo 1. Com isso, podemos afirmar que através do Algoritmo 2, todas as posições do vetor são incrementadas simultaneamente.

```
1  void inc_cpu(int *a,int N){
2     int idx;
3     for(idx=0;idx<N;idx++){
4        a[idx]=a[idx]+1;
5     }
6  }
7  void main(){
8     inc_cpu(a,N);
9  }
```

Algoritmo 1 – Incrementa *array* em CPU

```
1  __global__void inc_gpu(int *a_d,int N){
2     int idx=blockIdx.x*blockDim.x+threadIdx.x;
3     if(idx<N)
4        a_d[idx]=a_d[idx]+1;
5  }
6  void main(){
7     dim3 dimBLock(blocksize);
8     dim3 dimGrid(cell(N/(float)blocksize));
9     inc_gpu<<<dimGrid,dimBlock>>>(a_d,N);
10 }
```

Algoritmo 2 – Incrementa *array* em GPU

Nos dois exemplos apresentados, os dados de entrada são um vetor de inteiros e o tamanho deste vetor. É criada uma variável de índice idx que percorre todas as posições do vetor preenchendo-as com valores sequencias, variando de 0 até o tamanho do vetor.

No primeiro algoritmo o incremento é realizado de forma sequencial, através do uso da estrutura de repetição *for*. No segundo caso, são atribuídos variáveis da ferramenta CUDA que permitem incrementar, paralelamente, cada posição em uma *thread* da GPU.

4 Trabalhos Relacionados

Com intuito de identificar os principais trabalhos que se relacionam com este, foi elaboradass uma revisão bibliográfica, estabelecendo as similaridades, diferenças e possíveis tendências, baseadas no estado da arte.

Inicialmente, a modelagem do problema foi baseada no trabalho elaborado por Chen e Song (2009), onde é descrito um ambiente de *cross-docking* com restrições e particularidades similares à do presente trabalho. As definições de cálculo de *Lowerbound* e *GAP* também foram baseadas na referida obra e no projeto de Cota, Lira e Ravetti (2014), que tratam-se dos trabalhos que melhor se relacionam à proposta vigente neste trabalho.

Nossos resultados serão comparados com aqueles apresentados pelos trabalhos citados anteriormente, porém eles se diferem da presente proposta na forma como os autores construíram suas soluções, por meio de heurísticas híbridas e relaxações do problema, e pelo fato de que ambos autores não fazem uso de abordagens paralelas.

Boyer e Baz (2013) apresentaram uma compilação de estudos na literatura relacionando, de uma forma geral, a utilização da programação paralela em Processadores Gráficos (*GPUs*) na construção de algoritmos aplicados à Pesquisa Operacional. A principal contribuição do referido estudo para o presente trabalho é a exposição das mais recentes abordagens, classificadas em virtude dos algoritmos utilizados e apresentação de análise da performance computacional das soluções propostas. Entretanto, o referido trabalho trata-se de uma avaliação da literatura de propósito geral, não tendo foco nas especificidades de *Flowshop* ou *Cross-docking*.

De maneira similar, um grupo de pesquisadores da Universidade do Ceará (CARNEIRO et al., 2014) propôs o mapeamento sistemático da literatura relacionada a resolução de problemas de otimização combinatória através do uso de *GPUs*. Neste estudo estabeleceu-se as classes de algoritmos e os modelos de programação em GPU mais utilizados, além dos problemas de Pesquisa Operacional mais tratados através das referidas abordagens. O material em questão evidencia as estratégias paralelizadas com melhores resultados para problemas de sequenciamento, fato que será base para a elaboração da solução proposta pelo presente trabalho.

Um trabalho fundamentalmente importante para a construção desta proposta trata-se da publicação de Melab et al. (2012), onde propõe-se uma variação do algoritmo *Branch & Bound* paralelizado em GPU para a resolução do problema de sequenciamento do tipo *flowshop*. Descreve-se no referido trabalho a implementação de uma solução, utilizando o modelo de programação de paradigma paralelo por meio do padrão

CUDA, com objetivo de otimizar a troca de dados entre CPU e GPU na construção de quatro operadores básicos da solução: *branching*, *bounding*, *selection* e *elimination*, ou ramificação, vínculo, seleção e eliminação, respectivamente. Os autores ainda propõem um experimento onde são processadas instâncias de tamanho variado, no intuito de avaliar a performance computacional da solução proposta. Finalmente, compara-se o algoritmo criado com outras implementações do *Branch & Bound* da literatura, em função de seu tempo de execução. O referido trabalho mostra-se bastante similar à proposta do presente experimento, embora desconsidere as restrições de *Cross-docking* que almeja-se tratar.

Vallada, Ruiz e Framinan (2015) comparam qualitativamente diversas soluções, heurísticas e algoritmos utilizados na resolução dos problemas relacionados ao *Flowshop*. O intuito deste estudo é fornecer uma base de dados e parâmetros de *benchmark* (avaliação de desempenho) para a análise de soluções que prometem oferecer respostas viáveis para variações do problema abordado. O referido estudo, embora embasado em aplicações executadas somente em CPU, fornece importantes métricas que podem fundamentar a avaliação da performance da solução que o presente trabalho propõe-se a apresentar.

Belle, Valckenaers e Cattrysse (2012) apresentam uma visão geral sobre o tema *Cross-docking*, destacando os mais recentes trabalhos, até a data de publicação do referido estudo. Os autores classificam, cronologicamente, as soluções propostas na literatura, baseadas nas características e restrições do problema. Embora não apresente algoritmos diretamente relacionados às particularidades do problema que é tratado neste trabalho, o material compilado e indexado neste estudo é de grande valia como ampla revisão bibliográfica do tema em questão.

Shakeri et al. (2012) apresentam uma heurística para tratar o problema de sequenciamento de caminhões em um centro de *Cross-docking* com limitação de recursos. O estudo apresenta certas semelhanças com o problema que é tratado aqui, como a não preempção de recursos, objetivo de minimizar o makespan C_{max} e quantidade similar de máquinas. Mesmo notando que o referido trabalho difere em algumas restrições e características, nota-se que, a solução heurística proposta pelo autor pode embasar a construção de um algoritmo adaptado para o problema aqui abordado.

Não foram encontradas referências na literatura associadas especificamente à resolução de problemas de *Flowshop* com restrição de *Cross-docking* baseadas em programação paralela em *GPU*.

5 Metodologia

Com intuito de construir a solução proposta, validá-la e, posteriormente, publicá-la em um evento relacionado à área de aplicação da mesma, foram empregados os métodos descritos nas seções subsequentes, ordenados em função dos objetivos definidos.

5.1 Levantamento Bibliográfico

Foi realizado, em um primeiro momento, uma análise extensiva da literatura, de forma a compilar os materiais mais recentes, completos e com resultados mais relevantes. Assim, por meio deste procedimento, buscou-se apontar e analisar o estudo da arte e principais tendências dos tópicos relacionados aos objetos de estudo do presente trabalho, listados a seguir.

- Sequenciamento por *Flowshop* com restrições de *Cross-docking*;
- Heurísticas para solução do problema de *Cross-docking*:
  - Construção de heurísticas utilizando programação paralela;
- Programação paralela em GPU:
  - CUDA;
- Construção de algoritmos para solução de problemas de Pesquisa Operacional utilizando programação paralela em GPU.

O material obtido foi sintetizado no capítulo anterior. Os métodos e particularidades empregadas para resolução do problema abordado neste trabalho foram derivados daqueles mais promissores identificados.

5.2 Especificação do Problema

Conforme descrito anteriormente, o problema trata de uma estrutura de *cross-docking* que contempla várias docas de entrada e saída.

Sendo assim, a modelagem matemática do problema, que inclui as descrições das tarefas a serem processadas e em quais docas, conforme estabelecido na seção anterior, foi baseada na proposta de Chen e Song (2009) e Cota, Lira e Ravetti (2014).

5.2.1 Notação Matemática

A notação utilizada pode ser definida como:

Índices:

- i: estágio de processamento, $i \in \mathrm{I} = \{1,2\}$
- v_{ia_i}: máquina (doca) no estágio i, $v_{ia_i} \in V_i = \{v_{i1}, \ldots, v_{im_i}\}$, $a_i = 1, \ldots, m_i$
- j_{ie_i}: *job* (tarefa) a ser processado no estágio i, $j_{ie_i} \in J_i = \{j_{i1}, \ldots, j_{in_i}\}$, $e_i = 1, \ldots, n_i$

Parâmetros de Entrada:

- m_i: Número de máquinas (docas) no estágio i.
- n_i: Número de *jobs* no estágio i.
- p_{ie_i}: Tempo de processamento para o *job* e_i no estágio i.
- S_j: Conjunto de predecessores de j, que corresponde ao *job* $j_{2e_2} \in J_2, S_{e_2} \in S = \{S1, \ldots, S_{n_2}\}$

Variável de Decisão:

- C_{max}: Tempo necessário para processar todos os jobs em todos estágios, também chamado *makespan*.

5.2.2 Modelo Matemático

O problema abordado trata-se da minimização do tempo gasto (*makespan*), representado pela variável de decisão C_{max}, para alocar j_i *jobs* para processamento em m_i máquinas, sendo i o índice do estágio de processamento. Os *jobs* de saída possuem quantidade variável de predecessores, representados pela variável S_j, que são tarefas de entrada que devem ser processadas antes que seu sucessor seja analisado. Desta forma, o referido problema pode ser representado matematicamente, de acordo com Chen e Song (2009), como:

Parâmetro de Entrada:

- Q: Valor artificial muito grande, suficientemente maior que o *makespan*. Na prática, Q pode ser definido como $Q = \sum_{i \in e_I} \sum_{j_i e_i \in J_i P_{ie_i}}$.

Variáveis de Decisão:

- C_{ie_i}: Data de conclusão do *job* j_{ie_i} no estágio i;
- $x_{ia_ie_i}$: Variável binária que indica se o *job* j_{ie_i} foi alocado à máquina v_{ia_i} no estágio i;
- $y_{ie_if_i}$: Variável binária que indica se o *job* j_{ie_i} precede o *job* j_{if_i} no estágio i.

$$min\ C_{max} \tag{1}$$

Sujeito à:

$$\sum_{v_{ia_i} \in V_i} x_{ia_ie_i} = 1;\ \forall i \in I, j_{ie_i} \in J_i; \tag{2}$$

$$C_{1e_1} \geq p_{1e_1};\ \forall j_{1e_1} \in J_1; \tag{3}$$

$$C_{2e_2} \geq C_{1e_1} + p_{2e_2};\ \forall j_{2e_2} \in J_2, j_{ie_1} \in S_2, S_2 \in S; \\ C_{ie_i} + Q(2 + y_{ie_if_i} - x_{ia_ie_i} - x_{ia_ie_i}) \geq C_{if_i} + p_{ie_i}; \tag{4}$$

$$\forall i \in I, j_{ie_i}, j_{if_i} \in J_i, v_{ia_i} \in V_i; \\ C_{if_i} + Q(3 - y_{ie_if_i} - x_{ia_ie_i} - x_{ia_ie_i}) \geq C_{ie_i} + p_{if_i} \tag{5}$$

$$\forall i \in I, j_{ie_i}, j_{if_i} \in J_i, v_{ia_i} \in V_i; \tag{6}$$

$$C_{max} \geq C_{ie_i};\ \forall i \in I, j_{ie_i} \in j_i; \tag{7}$$

$$x_{ia_ie_i}, y_{ie_if_i} \in \{0, 1\};\ \forall i \in I, j_{ie_i}, j_{if_i} \in J_i, v_{ia_i} \in V_i. \tag{8}$$

Onde, (1) define a função objetivo, que é minimizar o *makespan* necessário para processar todos os *jobs*.

As restrições (2) garantem que cada *job* será atribuído a exatamente uma máquina ou doca.

As restrições (3) inferem que os *release dates* (tempo máximo para entrega de uma atividade ou tarefa) para os *jobs* de entrada, ou estágio 1, devem ser maiores ou iguais aos tempos de processamento dos respectivos *jobs*.

O conjunto de restrições (4) determinam que um *job* do segundo estágio, ou de saída, somente pode ser processado, assim que todos seus predecessores sejam processados.

Os conjuntos de restrições (5) e (6) garantem que diferentes *jobs* não sejam processados simultaneamente no mesmo processador, ou seja, não é possível alocar mais de um *job* no estágio i para a mesma máquina de i.

As restrições (7) definem o *makespan* máximo permitido para a conclusão de todos os *jobs*.

Finalmente, as restrições (8) determinam os domínios das variáveis de decisão especificadas previamente.

5.3 Tratamento do Problema

Para a resolução do problema, utilizou-se uma abordagem *multi-start*, fundamentada em duas heurísticas construtivas baseadas nas regras de LPT e LNS. As soluções inciais foram, posteriormente, refinadas utilizando as técnicas *Swap* e *Shift*, que buscam gerar novas soluções.

5.3.1 *Multi-start*

A abordagem *multi-start* pode ser definida como o processo de gerar uma ou mais soluções iniciais e melhorá-las por meio da aplicação sucessiva de heurísticas, buscas ou outras técnicas (MARTÍ, 2003).

A referida metodologia visa refinar um conjunto de soluções iniciais de forma que possa-se obter soluções ótimas globais para problemas complexos. A análise *multi-start* permite avaliar um espaço considerável de possibilidades em situações onde uma solução gulosa não é computacionalmente viável, em função do tempo gasto para executá-la (GLOVER, 2000).

De uma forma genérica, a metodologia *multi-start* pode ser ilustrada conforme o Algoritmo 3.

Algoritmo 3 *Multi-start*

```
1: procedimento MULTISTART
2:     F ← Constrói solução inicial;
3:     enquanto critério de parada não satisfeito faça
4:         S ← Gera nova solução;
5:         se (S < F) então
6:             F ← S;
7:         fim se
8:     fim enquanto
```

No presente trabalho, o procedimento adotado foi gerar duas soluções iniciais, por meio de duas heurísticas construtivas (LPT e LNS), que serão descritas a seguir, e refiná-las com as técnicas *Swap* e *Shift* na busca de melhores resultados.

5.3.2 LPT

Longest Processing Time (LPT) ou Maior Tempo de Processamento é uma regra ou algoritmo de aproximação que define que os *jobs* com maior tempo de processamento devem ser alocados preferencialmente nas máquinas que estiverem livres (CHEN, 1993),

O LPT tem um histórico de aplicação em pesquisas e na indústria, devido à sua simplicidade e relativa eficiência (MASSABÒ; PALETTA; RUIZ-TORRES, 2016).

No Algoritmo 4, apresenta-se um pseudocódigo para ilustrar a construção de uma sequência de *jobs* seguindo as regras do LPT. A cada iteração, busca-se o i-ésimo maior tempo de processamento, que é inserido na posição relativa da sequência final.

Algoritmo 4 LPT

```
1: função LPT(Jobs)
2:     para i variando de ← 0 até tamanho de Jobs - 1 faça
3:         sequencia[i] ← buscaMaiorTempo(Jobs, i);
4:     fim para
5: fim função
```

Foi feita uma adaptação do modelo LPT em função da variante do problema abordado, de forma a considerar as relações de precedência das máquinas de entrada em vista das máquinas de saída, assim, ordena-se os *jobs* de saída em função do maior tempo total resultante da soma dos tempos de seus predecessores (*jobs* de entrada).

A Figura 8 ilustra a execução do LPT, conforme praticado no presente trabalho. Neste exemplo, tem-se um total de 2 máquinas em cada estágio, um conjunto com 5 *jobs* de entrada $J_1 = \{j_0, ..., j_4\}$, um conjunto com 3 *jobs* de saída $J_2 = \{j_0, j_1, j_2\}$ e um conjunto de predecessores $S_2 = \{\{j_2, j_3, j_4\}, \{j_0, j_1\}, \{j_0\}\}$, onde cada subconjunto, em sua respectiva ordem, contém os predecessores do *job* de saída j_i. As cores no final de cada *job* são utilizadas para representar a relação de predecessores.

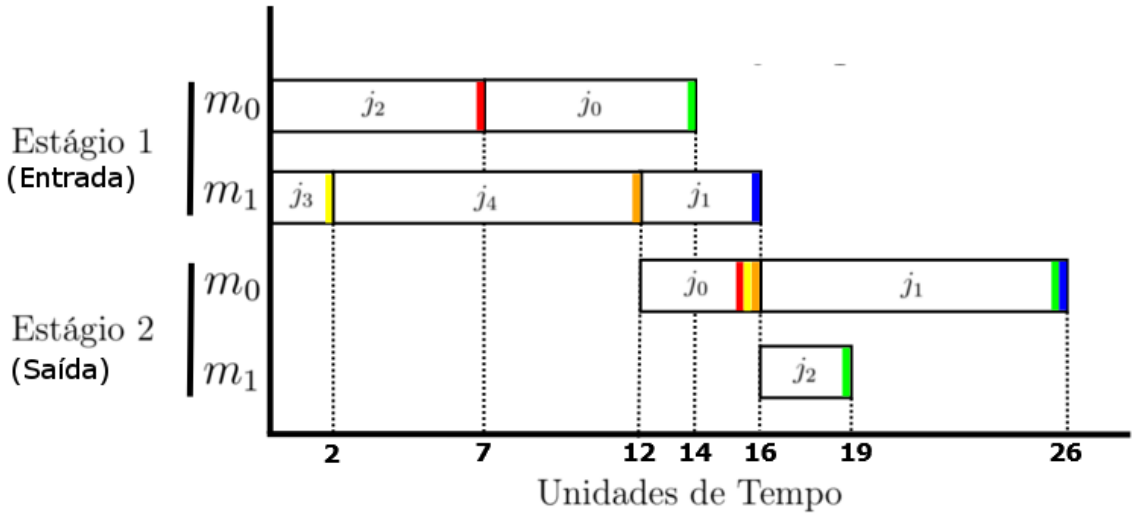

Figura 8 – Simulação para a execução do LPT

Fonte: Próprio autor

5.3.3 LNS

Largest Numbers of Successors First (LNS) ou Maior Número de Sucessores primeiro é uma regra de sequenciamento de tarefas que prevê que os elementos que possuem o maior número de sucessores devem ser alocados primeiramente. Desta forma, os *jobs* de entrada que são mais frequentes devem ter prioridade na sequência de alocação (PINEDO, 2015).

No Algoritmo 5, apresenta-se um pseudocódigo para ilustrar o sequenciamento de um se conjunto de *jobs*, de acordo com as propriedades do LNS. A cada iteração, busca-se analisar o número de sucessores que um *job* de entrada possui, ordenando-se a sequência de acordo com os *jobs* de entrada que possuem mais sucessores.

Algoritmo 5 LNS

```
1:  função LNS(JobsEntrada,JobsSaida)
2:      contador[TamanhoJobsEntrada];
3:      para i variando de ← 0 até tamanho de JobsEntrada - 1 faça
4:          para j variando de ← 0 até tamanho de JobsSaida - 1 faça
5:              se JobsEntrada[i] é predecessor de JobsSaida[j] então
6:                  contador[i] ++;
7:              fim se
8:          fim para
9:      fim para
10:     ordenaSequencia(contador[tamanho de JobsEntrada],jobsEntrada);
11: fim função
```

Neste trabalho, também foi feita uma adaptação da regra LNS, acrescentando dois passos adicionais:

1. Para os *jobs* de entrada que possuem o mesmo número de sucessores, opta-se por aquele cujo sucessor possui o menor número de predecessores, em outras palavras, aquele que faz um *job* de saída ser alocado o mais breve possível;

2. Além disso, caso haja dois sucessores com o mesmo número de predecessores opta-se pelo *job* de entrada com menor tempo de processamento.

A execução do LNS, conforme praticado neste trabalho, pode ser visualizada na Figura 9, supondo-se o mesmo exemplo de entrada utilizado na seção anterior (LPT).

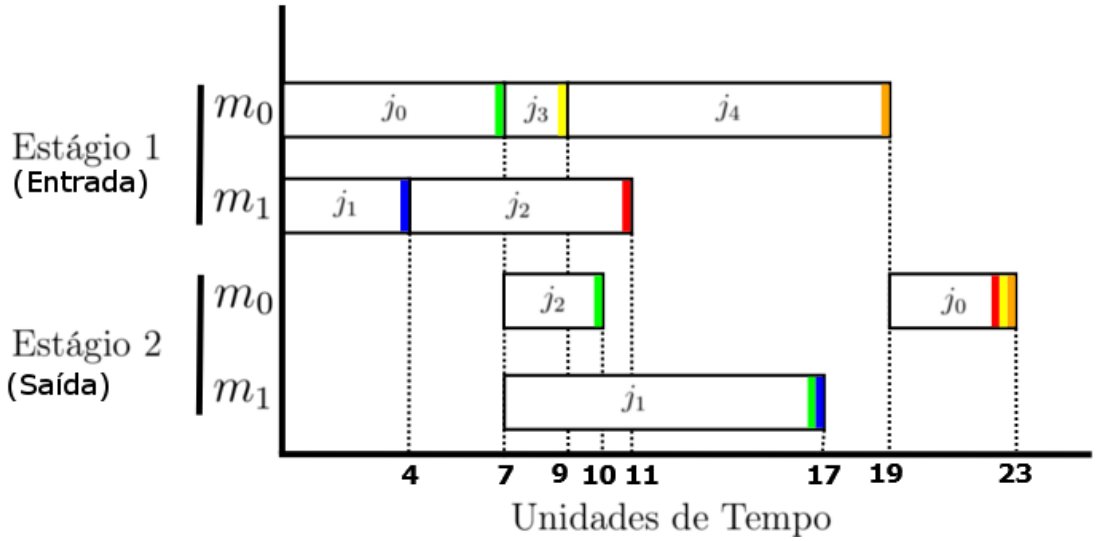

Figura 9 – Simulação para a execução do LNS

Fonte: Próprio autor

5.3.4 *Swap*

Swap pode ser definido como um modelo de algoritmo ou heurística de refinamento que constitui em combinar ou alterar os índices dos elementos de uma solução, no intuito de estabelecer novas soluções vizinhas e obter possíveis melhoras (KAO, 2008).

As referidas trocas ou *swaps* podem ser realizadas em níveis. Tal denominação refere-se a quantidade de elementos que serão trocados de uma vez. Por exemplo, quando um elemento é trocado com todos os demais, a troca é dita de nível 1. O *swap* pode ser incremental, quando o nível das trocas é gradualmente aumentado, no intuito de aumentar a variabilidade das vizinhanças. O funcionamento do processo é ilustrado através do Algoritmo 6. É válido notar que o processo de *Swap* visa buscar ótimos locais.

Algoritmo 6 *Swap*

```
1: procedimento SWAP
2:     S ← geraSoluçãoIncial(Jobs);
3:     para i variando de ← 0 até tamanho de S - 1 faça
4:         trocaElementos(S);
5:     fim para
```

Na implementação desenvolvida neste trabalho, foi aplicado o *swap* de nível 1 a partir das soluções resultantes das heurísticas construtivas (LPT e LNS), avaliando o *makespan* associado ao conjunto de *jobs* de saída e considerando o *First-improvement* ou o primeiro resultado com melhora. Além disso, durante o algoritmo, novas soluções correntes podem ser geradas, e a cada uma, a técnica *swap* é novamente aplicada.

A Figura 10 ilustra como seria uma iteração do processo de *Swap*, em nível 1, para os elementos de uma solução corrente. Nesta figura, a solução corrente é formada pelos *jobs* de saída $\{j_0, j_1, j_2\}$ e a técnica *swap* gera 3 novas sequências a partir da atual.

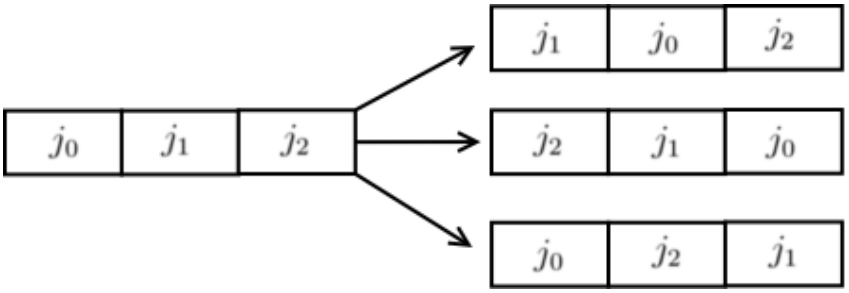

Figura 10 – Funcionamento do *Swap*

Fonte: Próprio autor

5.3.5 *Shift*

Shift é denominado uma heurística de refinamento ou técnica de perturbação, que pode ser descrita por meio do processo de deslocar um ou mais elementos de uma solução inicial, de forma em que haja alterações nas posições originais e tenha-se, ao final da iteração, uma solução inédita, razoavelmente distinta da original (KAO, 2008).

Na implementação desenvolvida, houve uma adaptação do modelo em que a técnica *shift* é aplicada iterativamente sempre que o *swap* não obtiver melhores *makespans*. Isto é, a partir de uma solução corrente, faz-se o *swap*, caso não houver melhora, aplica-se o *shift* e toma-se essa como a nova solução corrente, repetindo a aplicação de ambas técnicas (*swap* e *shift*). Se houver uma melhora, esta é tomada como a solução corrente e o processo é novamente repetido. É importante notar que o processo adaptado não testa todos os movimentos possíveis, mas apenas aqueles posteriores ao elemento (*job*) deslocado. O procedimento pode ser descrito genericamente por meio do Algoritmo 7.

Algoritmo 7 *Shift*

```
1: procedimento SHIFT
2:     S ← Conjunto Inicial;
3:     temp=S[0];
4:     para i ← variando de 0 até tamanho de S - 1 faça
5:         S[i] ← S[i]+1;
6:     fim para
7:     S[tamanho de S] ← temp;
```

A Figura 11 ilustra o resultado para o processo de *shift*, tal como praticado neste trabalho, aplicado duas vezes sobre uma solução corrente inicial Na situação apresentada, inicialmente a solução é formada pelos *jobs* de saída $\{j_0, j_1, j_2\}$.

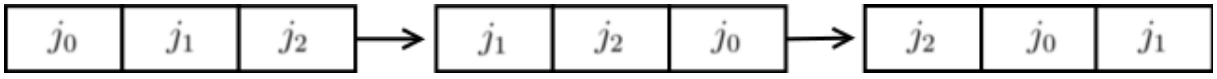

Figura 11 – Funcionamento do *Shift*

Fonte: próprio autor

5.4 Algoritmo CPU/GPU para a resolução do problema de *Cross-docking* híbrido

O método desenvolvido para a resolução do problema abordado neste trabalho é ilustrado por meio do Algoritmo 8.

Algoritmo 8 CCD

```
1: função CCD(Jobs)
2:     S ← LPT(Jobs) ou LNS(Jobs);
3:     existeMelhorSol ← verdadeiro;
4:     enquanto existeMelhorSol ou existeShift(S) faça
5:         i ← 0;
6:         existeMelhorSol ← falso;
7:         enquanto existeShift(S) faça
8:             S ← Shift(S, i);
9:             S' ← Swap(S) ou SwapGPU(S);
10:            se makespan(S') < makespan(S) então
11:                S ← S' ;
12:                existeMelhorSol ← verdadeiro;
13:                i ← 0;
14:                pare (break);
15:            fim se
16:            i ← i + 1;
17:        fim enquanto
18:    fim enquanto
19:    devolve S;
20: fim função
```

Como pode ser notado por meio da análise do pseudocódigo acima, as funcionalidades do algoritmo são realizadas para cada uma das soluções iniciais, geradas através das heurísticas construtivas LPT e LNS.

Além disso, no momento em que são geradas soluções vizinhas por meio da técnica *Swap*, foi desenvolvida uma implementação sequencial que utiliza apenas a CPU e uma implementação híbrida que utiliza tanto a CPU quanto a GPU para gerar as vizinhanças.

O critério de parada do algoritmo ocorre quando a combinação das técnicas *swap* e *shift* não surte efeitos positivos no *makespan* da solução analisada.

Os resultados obtidos por meio da execução do método proposto são exibidos na seção seguinte.

5.5 Dados de Entrada

Os dados de entrada foram obtidos a partir do trabalho de Cota, Lira e Ravetti (2014), cedidos pelo coautor do artigo Thiago Henrique Nogueira.

Como descrito no trabalho de Cota, Lira e Ravetti (2014), os dados são organizados em cinco grupos de máquinas, sendo três definidos de maneira determinística e os outros dois de forma probabilística. A distribuição é melhor apresentada abaixo:

- 2 máquinas nos dois estágios;
- 4 máquinas nos dois estágios;
- 10 máquinas nos dois estágios;
- Número de máquinas gerados por uma distribuição uniforme $Unif$ variando de 2 até 4, para cada estágio;
- Número de máquinas gerados por uma distribuição uniforme $Unif$ variando de 2 até 10, para cada estágio.

Em cada grupo de máquinas, o número de *jobs* no primeiro estágio varia de 20 a 70, em um intervalo de 10 em 10.

Para os jobs do segundo estágio, o número varia entre 80% e 120% do número de *jobs* do primeiro estágio, em cada instância.

De 20 a 70 *jobs*, foram avaliadas 300 instâncias.

A tabela a seguir ilustra a configuração dos dados de entrada.

Parâmetro	Notação	Valores testados
Número de jobs no 1º estágio	n_1	20,30,40,50,60,70
Número de jobs no 2º estágio	n_2	[16,24] para n_1 igual a 20
		[24,36] para n_1 igual a 30
		[32,48] para n_1 igual a 40
		[40,60] para n_1 igual a 50
		[48,72] para n_1 igual a 60
		[56,84] para n_1 igual a 70
Número de máquinas no estágio i	m_i	2,4,10, $Unif$[2, 4], $Unif$[2, 10]
Tempo de processamento	p_{ie_i}	$Unif$[10, 100]

Tabela 1 – Configuração dos dados de entrada

Fonte: (COTA; LIRA; RAVETTI, 2014)

Cada subgrupo de instâncias foi considerado como uma amostra, desta forma, obteve-se, a partir da análise dos mesmos, os valores para os piores, médios e melhores casos, conforme estabelecido através dos trabalhos de Chen e Song (2009) e Cota, Lira e Ravetti (2014).

5.6 Critérios de Avaliação

O desempenho computacional do algoritmo construído foi avaliado em função do tempo gasto para processar um número variável de instâncias, geradas segundo uma distribuição probabilística, configuradas conforme descrito na seção 5.5, cedidas pelos autores do trabalho, Cota, Lira e Ravetti (2014).

Devido a similaridade do tema e dos relevantes resultados obtidos, o documento desenvolvido por Cota, Lira e Ravetti (2014) foi utilizado como base de comparação para a qualidade dos resultados apresentados por meio do presente trabalho.

Os fatores utilizados para avaliação do algoritmo são fundamentados no calculo de GAP e *SpeedUp*.

5.6.1 *Lowerbound*

Antes de definir o GAP, é fundamental que o *Lowerbound* seja calculado, visto que o primeiro avalia a distância até esta métrica.

O *lowerbound* é um limite inferior definido para analisar a viabilidade da solução obtida, que deve ser, preferencialmente, minimamente superior a este fator.

A definição para o *lowerbound* utilizada neste artigo, elaborada por Cota, Lira e Ravetti (2014), considera o fato de que cada *job* de segundo estágio possui um tempo

médio de processamento em primeiro estágio, decorrente da relação de dependência em função de seus predecessores.

O *lowerbound*, segundo este princípio, pode ser calculado seguindo uma sequência de operações:

$$\overset{(1)}{p}_{e_2} = max\{max_{j_{ie_1}} \in S_{e_2} p_{1e_1}, \sum_{j_{1e_1} \in S_{e_2}} \frac{p1e_1}{m1}\} \tag{9}$$

$$\overset{(2)}{p}_{e_2} = p_{e_2} \tag{10}$$

$$\overset{(i)}{p}_q = \sum_{e_i=1}^{q} \overset{(i)}{pe}_i \tag{11}$$

$$LB = \frac{(\overset{(1)}{p}_{m_2}) + (\overset{(2)}{p}_{n_2})}{m_2} \tag{12}$$

A Equação 9 define o máximo entre o maior predecessor e a média dos tempos médios de processamento para cada *job* do estágio 2.

A Equação 10 define os tempos de processamento no segundo estágio, como os originais do problema, ou seja, eles não sofrem alteração, neste momento.

A Equação 11 refere-se ao somatório dos q menores tempos médios de processamento e dos q menores tempos de processamento do estágio 2.

Finalmente, na equação 12 é calculada a razão entre a soma das valores obtidos nos passos anteriores e a quantidade de máquinas em segundo estágio.

5.6.2 GAP

O GAP ou *Gain Average Percentage* ou Média Percentual de Ganho é dado pela distância entre o *makespan*, que é o resultado obtido, e o *Lowerbound*, ou limite inferior, definido anteriormente. O cálculo do GAP é feito através da seguinte fórmula:

$$GAP = \frac{(makespan - lowerbound)}{lowerbound} \tag{13}$$

O GAP apresenta a razão entre o resultado obtido e o resultado esperado, desta forma estabelecendo uma métrica para avaliação da qualidade dos resultados obtidos.

5.6.3 *SpeedUp*

O *SpeedUp* refere-se ao ganho de tempo, em vezes, de um elemento em relação a um referencial. No caso presente, analisou-se o ganho de tempo, em segundos, de pro-

cessamento de todas as instâncias pela GPU em relação a mesma operação feita pela CPU.

A fórmula que representa o *speedup* é:

$$SpeedUP = \frac{TempoCPU}{TempoGPU} \tag{14}$$

6 Resultados

Os algoritmos foram desenvolvidos na linguagem de programação C++, utilizando as bibliotecas da ferramenta CUDA. Os códigos foram compilados utilizando o compilador proprietário da NVIDIA, nvcc 8.0, componente do kit de desenvolvedor (SDK) CUDA 8.0, executado sob o compilador GNU gcc versão 5.1, utilizando o parâmetro (*flag*) de otimização -O3.

Os testes foram executados em um microcomputador DELL Inspiron 5557, arquitetura 64 bits, com processador Intel core I7 2,5 GHZ, 16 Gigabytes de memória RAM. A GPU utilizada trata-se de uma NVIDIA Geforce GTX TITAN X, com 12 Gigabytes de memória dedicada e 3072 CUDA cores. O Sistema Operacional utilizado foi o Linux Fedora 24 para os testes em CPU e o Linux Ubuntu 14.04 para os testes em GPU.

Os resultados computacionais obtidos são apresentados, posteriormente, em formato tabular onde as informações são indexadas em função da quantidade de *jobs* processados, número de máquinas, melhor GAP, GAP médio, pior GAP, tempo de processamento na CPU, tempo de processamento na GPU, Speedup (ganho em velocidade de processamento).

Em um primeiro momento, analisou-se, como proposto anteriormente, comparativamente os resultados obtidos apenas através da execução das heurísticas construtivas LPT e LNS com aqueles obtidos no trabalho de Cota, Lira e Ravetti (2014). Foi notado que, conforme pode ser visto na Tabela 1, os resultados obtidos inicialmente, antes do processo de refinamento, foram inferiores aos obtidos no referido trabalho. Isto ocorre pelo fato da qualidade das heurísticas utilizadas na referência serem comparativamente superiores às implementadas no presente trabalho, o que justifica a necessidade de refinamento do método proposto.

Na Tabela 1, os dados obtidos por meio da execução das heurísticas propostas estão indexados nas células e, entre parênteses, são exibidos os dados obtidos através do trabalho referenciado.

J_1	m_i	GAPs (%) GAP Medio	Melhor GAP	Pior GAP
	2	50,31 (33,8)	27,70 (14,15)	87,37 (70,54)
	4	56,25 (35,89)	33,69 (17,56)	91,27 (59,75)
20	10	47,91 (28,23)	24,52 (5,69)	71,13 (61,03)
	[2,4]	54,84 (36,48)	17,16 (9,02)	114,70 (84,26)
	[2,10]	56,08 (35,94)	8,60 (2,11)	132,89 (103,55)
	2	47,74 (36,07)	30,59 (17,96)	77,87 (53,14)
	4	53,97 (37,29)	36,63 (23,9)	82,99 (59)
30	10	55,97 (35,88)	32,63 (18,53)	83,10 (53,51)
	[2,4]	51,75 (36,48)	21,19 (9,02)	110,12 (84,26)
	[2,10]	57,77 (40,02)	9,58 (4,09)	130,44 (93,76)
	2	46,27 (37,53)	31,16 (23,77)	74,13 (57,4)
	4	50,63 (37,64)	34,99 (22)	77,78 (54,92)
40	10	60,81 (40,87)	36,70 (26)	84,88 (57,14)
	[2,4]	48,64 (38,18)	17,82 (10,61)	92,22 (79,1)
	[2,10]	56,07 (40,85)	10,18 (4,5)	114,48 (94,39)
	2	45,04 (38,17)	27,93 (23,65)	66,67 (58,48)
	4	48,67 (37,66)	31,50 (21,9)	68,97 (55,25)
50	10	61,74 (42,83)	38,89 (27,36)	81,40 (57,25)
	[2,4]	47,36 (38,69)	16,85 (12,44)	89,33 (76,62)
	[2,10]	54,38 (41,28)	9,16 (4,61)	117,93 (99,01)
	2	44,18 (38,7)	29,63 (26,46)	64,17 (56,3)
	4	46,91 (37,7)	32,11 (26,52)	62,86 (51,43)
60	10	58,67 (41,97)	43,64 (26,84)	82,43 (57,24)
	[2,4]	46,09 (38,74)	18,27 (11,83)	90,27 (81,55)
	[2,10]	52,13 (41,83)	9,98 (6,36)	109,53 (90,49)
	2	44,07 (39,69)	33,28 (27,81)	63,09 (55,89)
	4	46,15 (38,21)	35,36 (26,71)	62,84 (52,69)
70	10	56,12 (41,24)	38,99 (29,75)	75,83 (52,51)
	[2,4]	45,36 (39,3)	20,22 (16,39)	87,66 (75,35)
	[2,10]	50,66 (40,85)	11,36 (6,63)	103,61 (91,97)

Tabela 2 – Comparação dos melhores resultados das heurísticas construtivas e os obtidos no trabalho de Cota, Lira e Ravetti (2014).

Fonte: Próprio autor

Após a execução dos métodos construtivos e a coleta dos dados inciais, executou-se as técnicas propostas para o refinamento da solução, *swap* e *shift*.

Partindo-se dos resultados iniciais, obtidos por meio da execução das heurísticas construtivas, verifica-se que houve significante melhoria em função dos melhores GAPs obtidos, quando comparamos a solução final do algoritmo proposto, após o processo de melhoria utilizando as técnicas mencionadas acima, com aquela verificada inicialmente.

A Figura 12 ilustra a diferença dos valores de GAPs em função das configurações de máquinas utilizadas e os números de *jobs* processados.

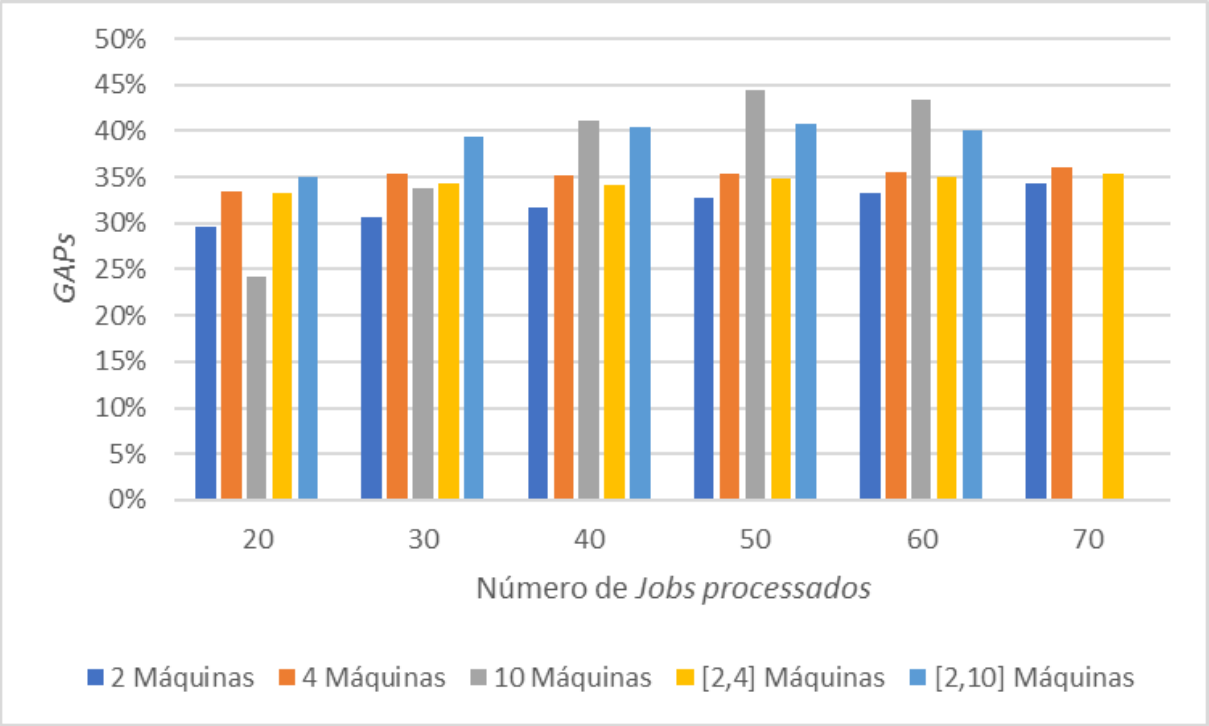

Figura 12 – Comparação dos GAPs médios da solução final em função da configuração de máquinas

Fonte: Próprio autor

A comparação dos GAPs obtidos por meio da execução das heurísticas construtivas e do modelo refinado, classificados por configuração de máquinas, pode ser ilustrada através das Figuras 13, 14, 15, 16 e 17.

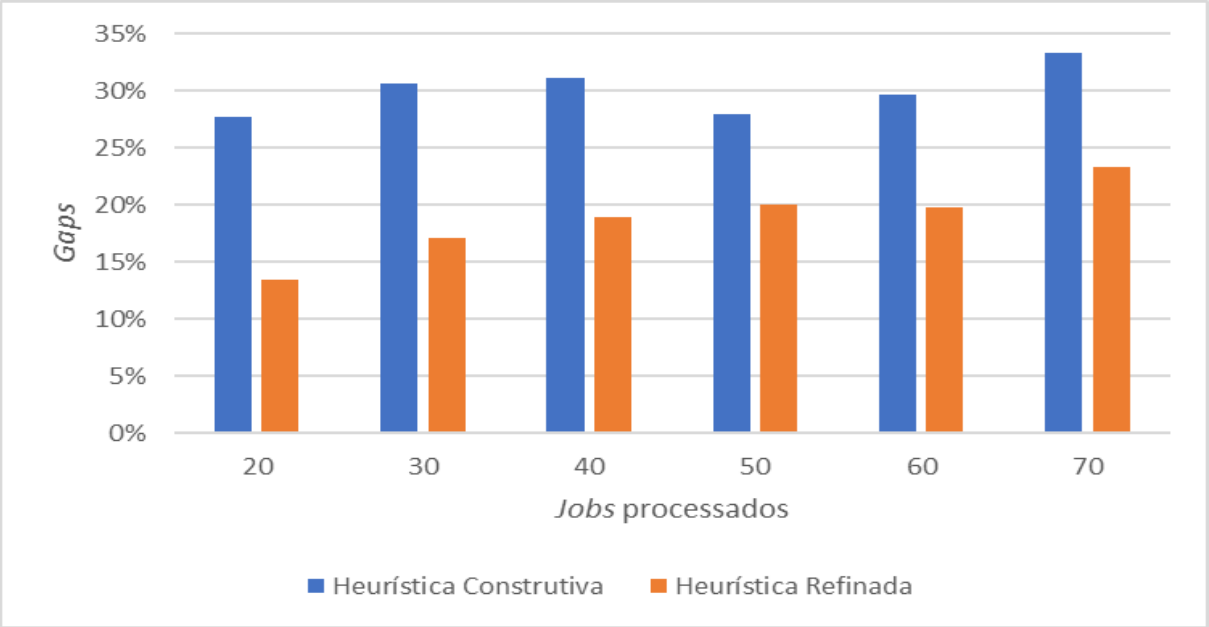

Figura 13 – Comparação dos melhores GAPs das soluções refinadas e construtivas para 2 máquinas

Fonte: Próprio autor

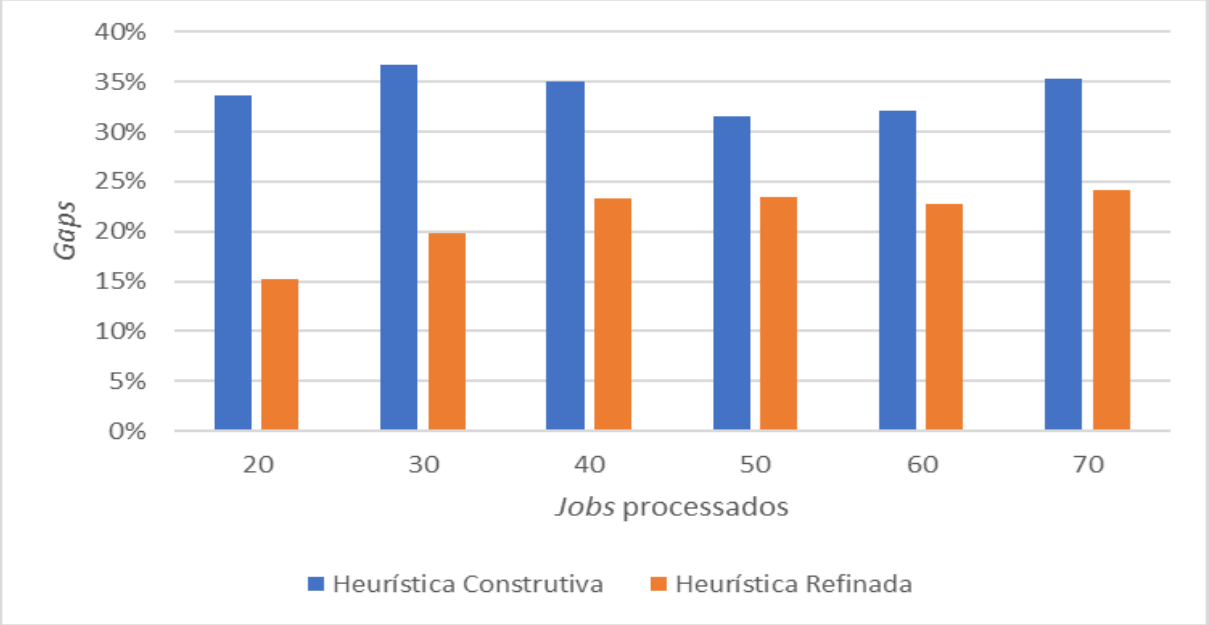

Figura 14 – Comparação dos melhores GAPs das soluções refinadas e construtivas para 4 máquinas

Fonte: Próprio autor

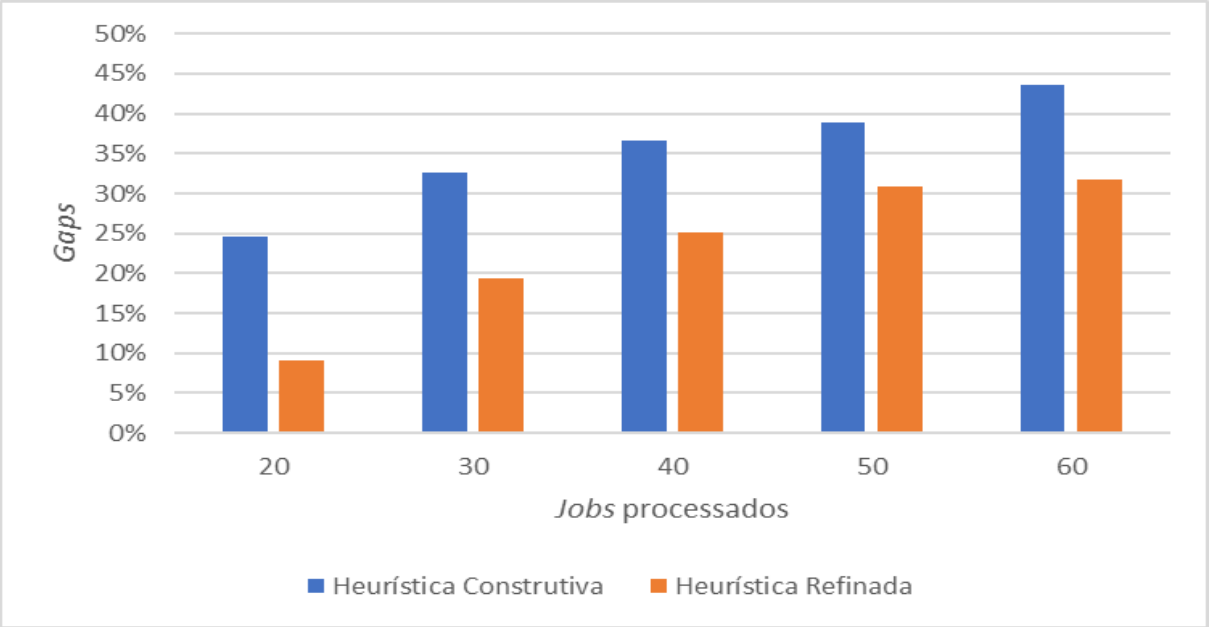

Figura 15 – Comparação dos melhores GAPs das soluções refinadas e construtivas para 10 máquinas

Fonte: Próprio autor

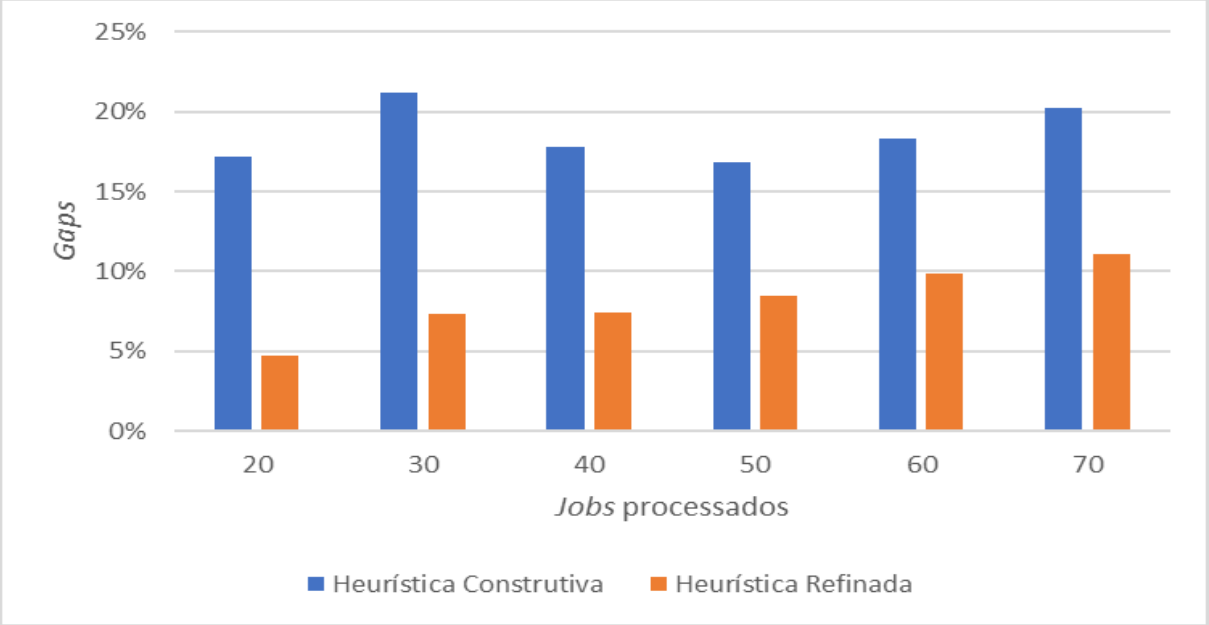

Figura 16 – Comparação dos melhores GAPs das soluções refinadas e construtivas para [2,4] máquinas

Fonte: Próprio autor

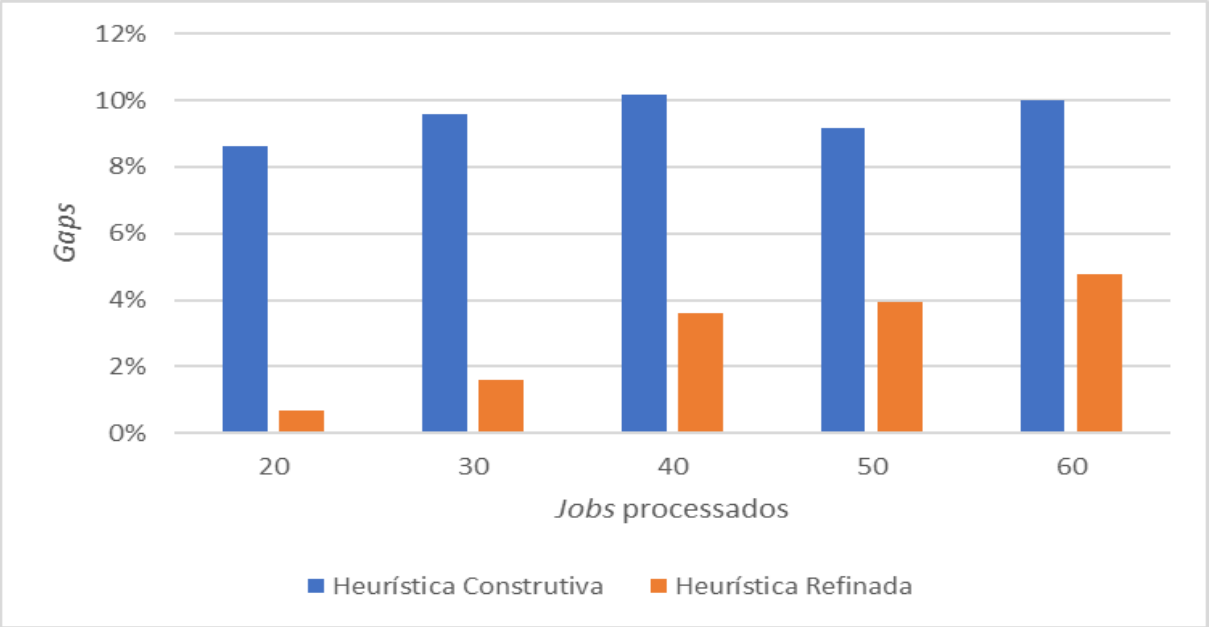

Figura 17 – Comparação dos melhores GAPs das soluções refinadas e construtivas para [2,10] máquinas

Fonte: Próprio autor

As Figuras 18, 19, 20, 21 e 22 ilustram a comparação entre o desempenho da solução sequencial e da solução híbrida, de acordo com a configuração de máquinas.

Outro resultado obtido foi a comparação entre o desempenho de uma solução sequencial utilizando CPU e uma solução híbrida, que empregou CPU e GPU. Foi notado, por meio da execução dos testes que, para a quantidade de máquinas variando de 20 a 40, a solução sequencial obteve ampla vantagem, em função de seu tempo de processamento.

Para 50 máquinas, os resultados, de desempenho computacional, obtidos comparando as duas soluções foram virtualmente similares.

Finalmente, para a quantidade de máquinas variando de 60 a 70, foi visto que a solução híbrida obteve considerável vantagem, em relação ao tempo gasto pela solução sequencial.

Portanto, verificou-se que a abordagem utilizando CPU é visivelmente mais eficiente em relação aquela que faz o uso da GPU para quantidades inferiores de máquinas. Entretanto, para valores maiores em número de máquinas, a abordagem híbrida, complementada pelo uso de GPU, obteve nítida vantagem, em função do tempo gasto para processar as instâncias propostas, que alcançou uma proporção de superioridade de até 2 vezes (*SpeedUp*).

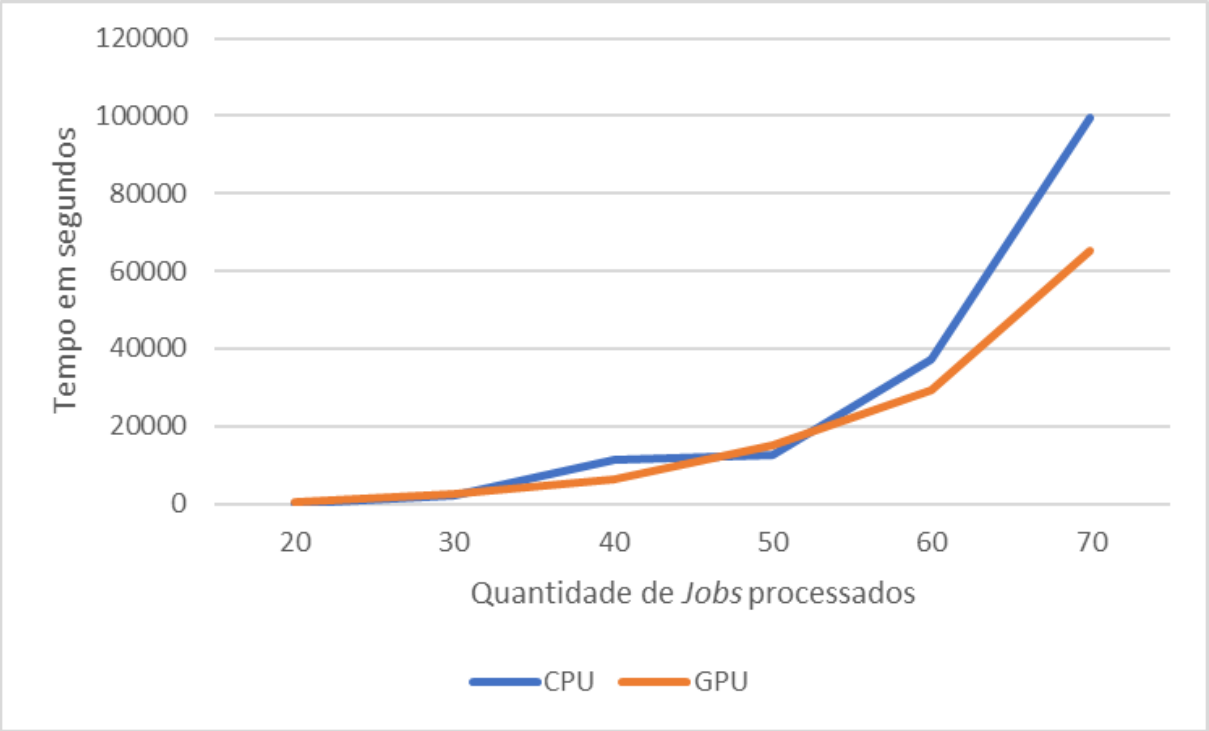

Figura 18 – Comparação do Desempenho da CPU e GPU para 2 máquinas

Fonte: Próprio autor

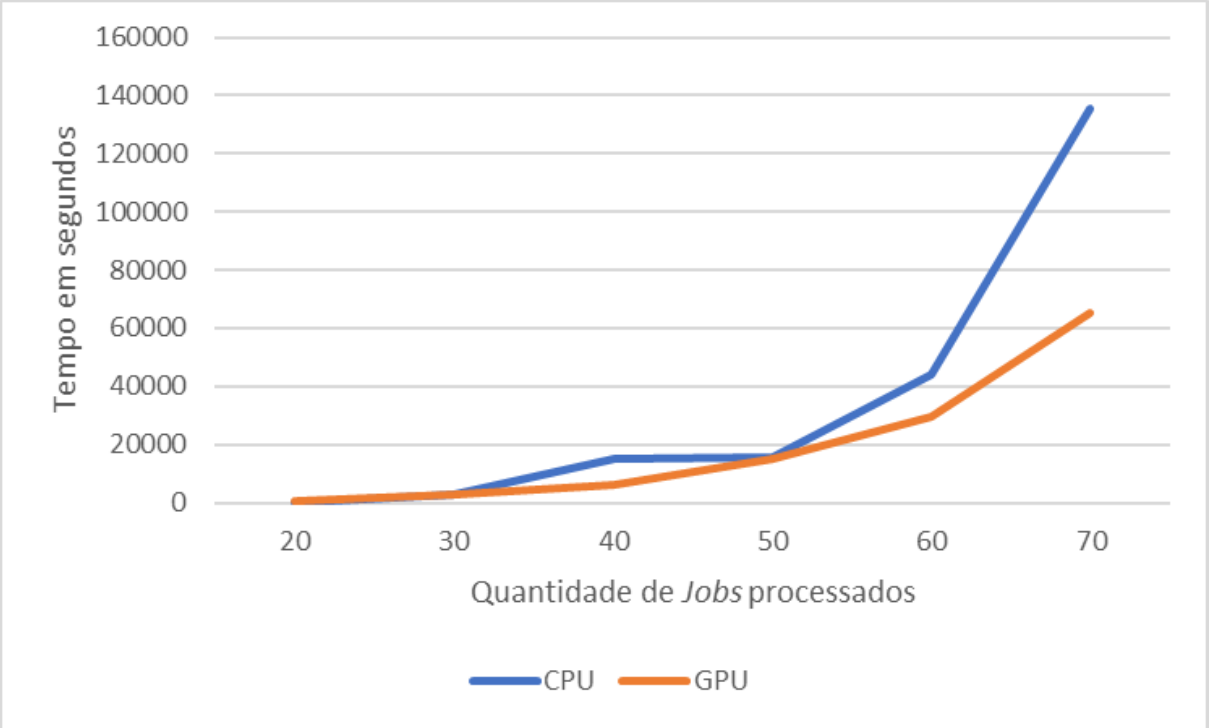

Figura 19 – Comparação do Desempenho da CPU e GPU para 4 máquinas

Fonte: Próprio autor

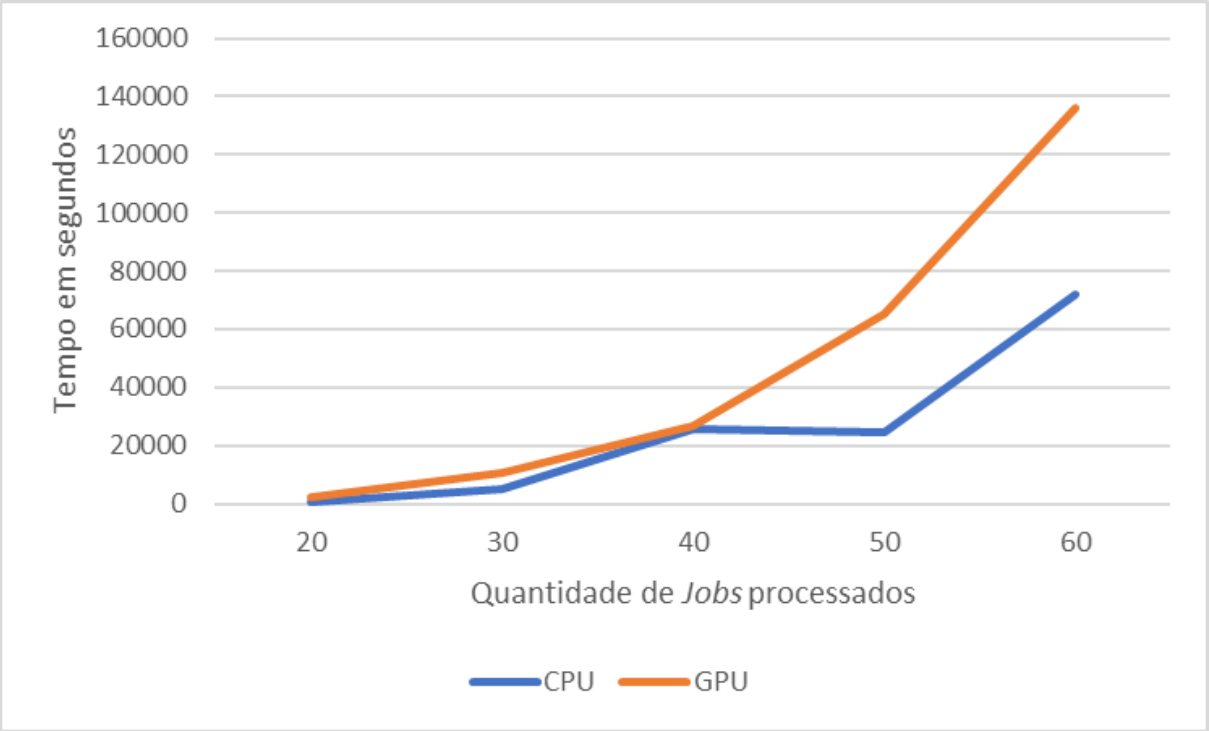

Figura 20 – Comparação do Desempenho da CPU e GPU para 10 máquinas

Fonte: Próprio autor

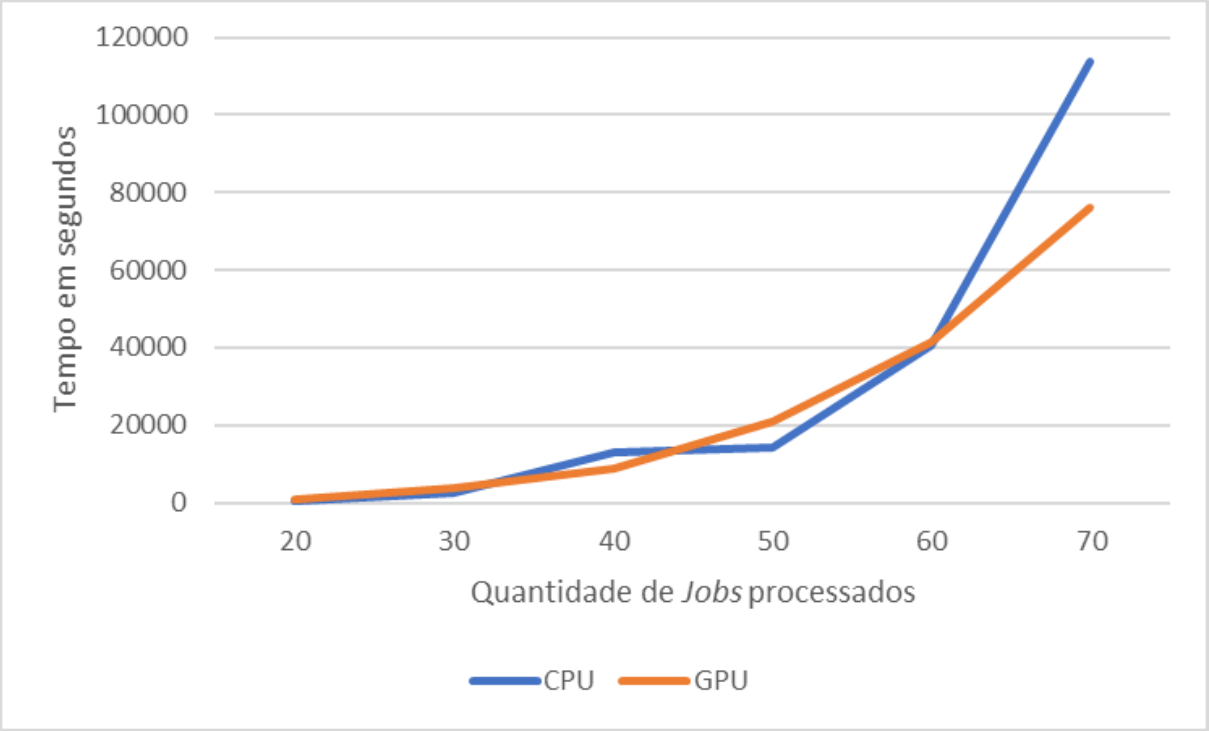

Figura 21 – Comparação do Desempenho da CPU e GPU para [2,4] máquinas

Fonte: Próprio autor

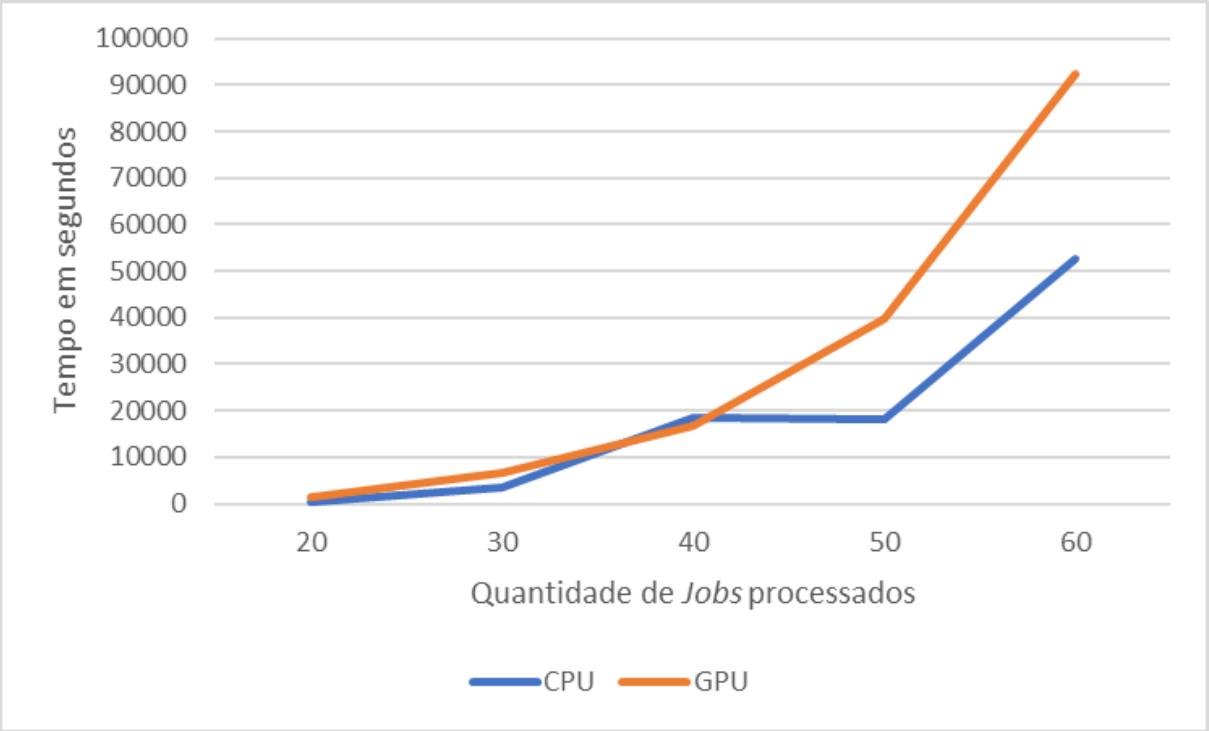

Figura 22 – Comparação do Desempenho da CPU e GPU para [2,10] máquinas

Fonte: Próprio autor

O algoritmo sequencial foi otimizado, por meio do compilador GCC, utilizando o parâmetro -O3, além disso, o recurso de processamento em GPU, verificado através da execução da solução híbrida, não foi otimamente utilizado, de forma que o valor médio da utilização foi de cerca de 25%.

Desta forma, verificou-se que a utilização conjunta da CPU e GPU obteve resultados promissores, para a resolução de um problema que é dito, na literatura, como extremamente dependente, em função das relações de precedência entre as máquinas de diferentes estágios (BELLE; VALCKENAERS; CATTRYSSE, 2012).

A Tabela 3 ilustra os resultados finais obtidos. Em parênteses estão os resultados obtidos pela execução, de uma vez por instância, das heurísticas construtivas e os demais são aqueles relativos à execução das técnicas de refinamento dos valores iniciais. Por fim, tem-se a comparação entre os tempos gastos pela CPU e GPU para processar as instâncias e o ganho, em vezes, da solução híbrida em função da solução sequencial.

J_1	m_i	GAPs (%)			Tempos (segundos)		SpeedUp
		GAP Médio	Melhor GAP	Pior GAP	Tempo CPU	Tempo GPU	
	2	29,62 (50,31)	13,49 (27,70)	61,56 (87,37)	208,21	523,07	0,40x
	4	33,37 (56,25)	15,25 (33,69)	60,98 (91,27)	287,40	957,30	0,30x
20	10	24,18 (47,91)	9,09 (24,52)	48,21 (71,13)	486,90	2109,57	0,23x
	[2,4]	33,19 (54,84)	4,71 (17,16)	99,21 (114,70)	242,02	734,07	0,33x
	[2,10]	35,02 (56,08)	0,67 (8,60)	119,41 (132,89)	329,69	1308,66	0,25x
	2	30,68 (47,74)	17,04 (30,59)	51,87 (77,87)	2113,85	2620,05	0,81x
	4	35,34 (53,97)	19,77 (36,63)	59,22 (82,99)	2780,18	4659,47	0,60x
30	10	33,86 (55,97)	19,40 (32,63)	55,81 (83,10)	4859,57	10812,2	0,45x
	[2,4]	34,25 (51,75)	7,31 (21,19)	86,10 (110,12)	2403,14	3592,16	0,67x
	[2,10]	39,43 (57,77)	1,61 (9,58)	104,78 (130,44)	3438,03	6714,82	0,51x
	2	31,70 (46,27)	18,93 (31,16)	53,19 (74,13)	11148,1	6272,23	1,78x
	4	35,22 (50,63)	23,32 (34,99)	56,87 (77,78)	14802,7	11514,2	1,29x
40	10	41,03 (60,81)	25,14 (36,70)	60,71 (84,88)	25694,8	26885,6	0,96x
	[2,4]	34,10 (48,64)	7,42 (17,82)	78,84 (92,22)	13087,2	8981,03	1,46x
	[2,10]	40,37 (56,07)	3,59 (10,18)	98,81 (114,48)	18350	16674,4	1,10x
	2	32,80 (45,04)	19,97 (27,93)	54,22 (66,67)	12496,1	14858,9	0,84x
	4	35,37 (48,67)	23,51 (31,50)	51,15 (68,97)	15596,8	27104,5	0,58x
50	10	44,37 (61,74)	30,79 (38,89)	58,89 (81,40)	24771,6	65180,5	0,38x
	[2,4]	34,76 (47,36)	8,47 (16,85)	74,52 (89,33)	14194,9	20982	0,68x
	[2,10]	40,82 (54,38)	3,95 (9,16)	105,05 (117,93)	18254,1	39614,3	0,46x
	2	33,20 (44,18)	19,76 (29,63)	50,89 (64,17)	37104	29299,2	1,27x
	4	35,53 (46,91)	22,70 (32,11)	49,46 (62,86)	43963,5	53728,2	0,82x
60	10	43,40 (58,67)	31,75 (43,64)	59,17 (82,43)	72033,6	136285	0,53x
	[2,4]	34,93 (46,09)	9,83 (18,27)	80,54 (90,27)	40561,9	41562,9	0,98x
	[2,10]	40,15 (52,13)	4,78 (9,98)	97,06 (109,53)	52632,1	92384,6	0,57x
	2	34,39 (44,07)	23,29 (33,28)	53,28 (63,09)	99603,1	65314,6	1,52x
	4	36,02 (46,15)	24,09 (35,36)	53,02 (62,84)	135258	102069	1,33x
70	[2,4]	35,42 (45,36)	11,04 (20,22)	73,04 (87,66)	113790	76102	1,49x

Tabela 3 – Resultados obtidos por meio das técnicas de refinamento *swap* e *shift* a partir de soluções iniciais geradas pelas heurísticas construtivas.

Fonte: Próprio autor

7 Conclusão

Cross-docking é uma classe de problemas de sequenciamento que possui uma relação de precedência entre as máquinas de diferentes estágios, que o torna complexo e dependente. Por ser um problema de complexidade exponencial, as abordagens existentes para solucioná-lo não garantem resultados próximos aos ótimos em tempo viável, o que justifica a necessidade de desenvolver e refinar heurísticas que sejam capazes de oferecer soluções satisfatórias para este problema.

A particularidade do problema abordado neste trabalho trata-se do ambiente de *Cross-docking* com múltiplas docas e quantidade distinta de máquinas em estágios diferentes, fator que torna o problema ainda mais restrito. As abordagens existentes na literatura para essa especificação são bastante escassas. Logo, este trabalho foi embasado nas referências de Cota, Lira e Ravetti (2014) e Chen e Song (2009).

A abordagem utilizada, baseou-se na técnica de *multi-start*, fundamentada em duas heurísticas construtivas, LPT e LNS, cujos resultados iniciais foram, posteriormente, refinados com auxílio dos procedimentos de busca local, *Swap* e *Shift*.

Os resultados iniciais, obtidos pro meio da execução das heurísticas construtivas, foram inferior àqueles tomados como referência, o que justificou a necessidade do refinamento. Através do processo de melhoria, os resultados finais mostraram-se consideravelmente superiores aos obtidos inicialmente, o que atesta a qualidade das técnicas empregadas na resolução apresentada por este trabalho.

Por meio da técnica de programação híbrida, sustentada pela utilização conjunta da CPU e GPU, provou-se, que resultados promissores foram obtidos, evidenciando o potencial da metodologia CUDA para a solução de problemas de pesquisa operacional, em que o processamento paralelo de dados provê uma boa alternativa.

Finalmente, propõe-se, como trabalhos futuros a escrita de um artigo para documentar o que foi obtido por meio do presente trabalho e apresentar a contribuição do mesmo para a comunidade acadêmica. Pretende-se, também, realizar a otimização do uso da plataforma CUDA para a resolução do problema proposto, utilizar a referida metodologia para resolução de outras variantes do problema de *Cross-docking* e aplicar a técnica *multi-start*, sustentada pela programação paralela em GPU, para a análise de outros problemas que oferecem amplo escopo de soluções.

Referências

ALMASI, G. S.; GOTTLIEB, A. **Highly Parallel Computing**. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1989. Citado na página 18.

APTE, U. M.; VISWANATHAN, S. Effective cross docking for improving distribution efficiencies. **International Journal of Logistics Research and Applications**, v. 3, n. 3, p. 291–302, 2000. Citado na página 16.

BARNEY, B. Introduction to parallel computing. **Lawrence Livermore National Laboratory**, v. 6, n. 13, p. 10, 2010. Citado na página 18.

BELLE, J. V.; VALCKENAERS, P.; CATTRYSSE, D. Cross-docking: State of the art. **Omega**, Elsevier, v. 40, n. 6, p. 827–846, 2012. Citado 4 vezes nas páginas 12, 17, 23 e 45.

BOURGOIN, M.; CHAILLOUX, E.; LAMOTTE, J.-L. Efficient abstractions for gpgpu programming. **International Journal of Parallel Programming**, v. 42, n. 4, p. 583–600, 2014. Citado na página 19.

BOYER, V.; BAZ, D. E. Recent Advances on GPU Computing in Operations Research. **27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum**, 2013. Citado 3 vezes nas páginas 10, 20 e 22.

CARNEIRO, T.; JÚNIOR, F. H. de C.; ARRUDA, N. G. P. B.; PINHEIRO, A. B. Um levantamento na literatura sobre a resolução de problemas de otimização combinatória através do uso de aceleradores gráficos. **Proceedings of the XXXV Iberian Latin-American Congress on Computational Methods in Engineering**, 2014. Citado na página 22.

CHEN, B. A note on lpt scheduling. **Operations Research Letters**, v. 14, n. 3, p. 139 – 142, 1993. ISSN 0167-6377. Disponível em: <http://www.sciencedirect.com/science/article/pii/016763779390024B>. Citado na página 27.

CHEN, F.; SONG, K. Minimizing makespan in two-stage hybrid cross docking scheduling problem. **Computers & Operations Research**, Elsevier, v. 36, n. 6, p. 2066–2073, 2009. Citado 7 vezes nas páginas 10, 17, 22, 24, 25, 34 e 47.

COTA, P. M.; LIRA, E. G.; RAVETTI, M. G. O problema de sequenciamento de caminhões em centros de crossdocking com múltiplas docas. **XLVI Simpósio Brasileiro De Pesquisa Operacional**, 2014. Citado 8 vezes nas páginas 5, 22, 24, 33, 34, 37, 38 e 47.

GLOVER, F. Multi-start and strategic oscillation methods—principles to exploit adaptive memory. In: **Computing Tools for Modeling, Optimization and Simulation**. [S.l.]: Springer, 2000. p. 1–23. Citado na página 27.

KAO, M.-Y. **Encyclopedia of algorithms**. [S.l.]: Springer Science & Business Media, 2008. Citado 2 vezes nas páginas 30 e 31.

KIM, B. S.; JOO, C. M. Scheduling Trucks in Multi-Door Cross Docking Systems : An Adaptive Genetic Algorithm with a Dispatching Rule. v. 32, n. 3, p. 1–20, 2015. Citado na página 17.

KINNEAR, E. Is there any magic in cross-docking? **Supply Chain Management: An International Journal**, v. 2, n. 2, p. 49–52, 1997. Citado 3 vezes nas páginas 10, 14 e 17.

KÜÇÜKOĞLU, İ.; ÖZTÜRK, N. A hybrid meta-heuristic algorithm for vehicle routing and packing problem with cross-docking. **Journal of Intelligent Manufacturing**, p. 1–17, 2015. Citado na página 17.

KUMAR, V.; GRAMA, A.; GUPTA, A.; KARYPIS, G. **Introduction to parallel computing: design and analysis of algorithms**. [S.l.]: Benjamin/Cummings Redwood City, CA, 1994. v. 400. Citado 2 vezes nas páginas 18 e 19.

KUNDAKCI, N.; KULAK, O. Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. **Computers& Industrial Engineering**, 2016. Citado na página 13.

LIAO, T.; EGBELU, P.; CHANG, P. Simultaneous dock assignment and sequencing of inbound trucks under a fixed outbound truck schedule in multi-door cross docking operations. **International Journal of Production Economics**, v. 141, n. 1, p. 212 – 229, 2013. Meta-heuristics for manufacturing scheduling and logistics problems. Citado na página 17.

MARTÍ, R. Multi-start methods. In: _____. **Handbook of Metaheuristics**. Boston, MA: Springer US, 2003. p. 355–368. ISBN 978-0-306-48056-0. Disponível em: <http://dx.doi.org/10.1007/0-306-48056-5_12>. Citado na página 27.

MASSABÒ, I.; PALETTA, G.; RUIZ-TORRES, A. J. A note on longest processing time algorithms for the two uniform parallel machine makespan minimization problem. **Journal of Scheduling**, v. 19, n. 2, p. 207–211, 2016. ISSN 1099-1425. Disponível em: <http://dx.doi.org/10.1007/s10951-015-0453-x>. Citado na página 28.

MELAB, N.; CHAKROUN, I.; MEZMAZ, M.; TUYTTENS, D. A GPU-accelerated Branch-and-Bound Algorithm for the Flow-Shop Scheduling Problem. **14th IEEE International Conference on Cluster Computing, Cluster'12**, 2012. Citado na página 22.

MITTAL, S.; VETTER, J. S. A survey of cpu-gpu heterogeneous computing techniques. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 47, n. 4, p. 69:1–69:35, jul. 2015. Citado na página 19.

NOGUEIRA, A. d. S. **Logística Empresarial: Uma visão local com pensamento globalizado**. [S.l.]: São Paulo: Atlas, 2012. Citado na página 15.

NVIDIA. **Cuda c programming guide**. [S.l.: s.n.], 2015. v. 7.5. Citado na página 20.

PINEDO, M. **Scheduling**. [S.l.]: Springer, 2015. Citado na página 29.

PINEDO, M. L. **Scheduling: theory, algorithms, and systems**. [S.l.]: Springer Science & Business Media, 2008. Citado na página 12.

SHAKERI, M.; LOW, M. Y. H.; TURNER, S. J.; LEE, E. W. A robust two-phase heuristic algorithm for the truck scheduling problem in a resource-constrained crossdock. **Computers & Operations Research**, v. 39, n. 11, p. 2564 – 2577, 2012. Citado 2 vezes nas páginas 17 e 23.

STALK, G.; EVANS, P.; SHULMAN, L. E. Competing on capabilities: the new rules of corporate strategy. **Harvard business review**, v. 70, n. 2, p. 57–69, 1991. Citado na página 17.

STEPHAN, K.; BOYSEN, N. Cross-docking. **Journal of Management Control**, v. 22, n. 1, p. 129–137, 2011. Citado na página 14.

VALLADA, E.; RUIZ, R.; FRAMINAN, J. M. New hard benchmark for flowshop scheduling problems minimising makespan. **European Journal of Operational Research**, Elsevier, v. 240, n. 3, p. 666–677, 2015. Citado 3 vezes nas páginas 10, 13 e 23.