UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
INSTITUTO DE BIOCIÊNCIAS, LETRAS E CIÊNCIAS EXATAS

IGOR STEFANI BUTTARELLO

INVESTIGAÇÃO SOBRE O USO DE GPUS EM PLANEJAMENTO DE PRODUÇÃO

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

São José do Rio Preto
2014

IGOR STEFANI BUTTARELLO

INVESTIGAÇÃO SOBRE O USO DE GPUS EM PLANEJAMENTO DE PRODUÇÃO

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

Orientador:
Prof. Dr. Aleardo Manacero Jr.

São José do Rio Preto
2014

Buttarello, Igor Stefani.
Investigação sobre o uso de GPUS em planejamento de produção / Igor Stefani Buttarello. -- São José do Rio Preto, 2014
34 f. : il., tabs.

Orientador: Aleardo Manacero Junior
Trabalho de conclusão de curso (Bacharelado – Ciência da Computação) – Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas

1. Computação. 2. Programação paralela (Computação) 3. Unidades de processamento gráfico. 4. CUDA (Arquitetura de computador) 5. Planejamento da produção. 6. Processos de fabricação - Modelos matemáticos. 7. Algoritmos paralelos. I. Manacero Junior, Aleardo. II. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. III. Título.
CDU – 518.721

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

IGOR STEFANI BUTTARELLO

INVESTIGAÇÃO SOBRE O USO DE GPUS EM PLANEJAMENTO DE PRODUÇÃO

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

_____ _____

Prof. Dr. Aleardo Manacero Jr. (orientador) Igor Stefani Buttarello (aluno)

Banca examinadora:
Prof. Dr. Silvio Alexandre de Araujo
Profa. Dra. Luciana Pavani de Paula Bueno

São José do Rio Preto
2014

Dedico este trabalho aos meus pais e meu irmão que estiveram ao meu lado apoiando e aconselhando durante todos estes anos para que eu nunca desistisse dos meus ideais, sempre demonstrando lindos exemplos de amor e carinho.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus por me abençoar com a linda família e os poucos, porém leais, amigos que estiveram ao meu lado durante essa jornada de desafios, lutas, quedas e conquistas... Com certeza, muitas conquistas.

Agradeço a toda minha família, em especial minha mãe, meu pai e meu irmão por serem as pessoas mais importantes em minha vida e que unidos sempre compartilharam todos os momentos, alegres ou tristes, da minha vida.

Agradeço à minha namorada Alessandra por ter me apoiado em um dos momentos mais difíceis da minha vida, bem como por ser paciente e compreensiva com meus estudos e busca por ideais.

Agradeço ao prof. Aleardo (orientador) e à profa. Renata por tratarem a arte de lecionar com exímia habilidade, transmitindo o conhecimento da melhor qualidade possível e, também, por terem me acolhido no Grupo de Sistemas Paralelos e Distribuídos (GSPD) onde pude estar em contato direto e aprender muito sobre a verdadeira computação.

Agradeço a todos os meus amigos, em especial ao Tim (vulgo Renato), Birruga (vulgo Rodrigo), Gabriel, Cabeça (vulgo Hector) e Diogo, os quais marcaram presença em minha vida. Com eles aprendi muito e compartilhei momentos de amizade, os quais poucas pessoas têm o privilégio de vivenciar.

“With great power comes great responsibility!”

Uncle Ben

RESUMO

Este trabalho propõe um estudo quanto ao uso de técnicas de processamento paralelo na execução de algoritmos de planejamento de produção, visando, desta forma, tanto apresentar melhorias em um sistema de manufatura quanto buscar respostas mais rápidas e precisas quando da ocorrência de eventos estocásticos que afetem o escalonamento da linha de produção de uma fábrica. O processamento paralelo se dará pelo uso de GPU's (Graphics Processing Units) desenvolvidas com a arquitetura "CUDA", a qual habilita tais dispositivos a processar dados de forma paralela usando os núcleos do processador gráfico ("CUDA cores"). A linguagem de programação utilizada nas GPUs habilitadas para CUDA é o "CUDA C" que, na prática, trata-se da tradicional linguagem "C" com um adicional de funções para controle de programação paralela massiva nos dispositivos gráficos em questão. Haverá também a elaboração de testes, os quais apresentarão as vantagens do paralelismo fornecido pelas GPUs quando da execução dos códigos de algoritmos (heurísticos) de busca, em contraste com a execução sequencial dos mesmos.

Palavras-chave: planejamento de produção – escalonamento em sistemas de manufatura – GPUs CUDA – processamento paralelo – escalonador CUDA

ABSTRACT

This work proposes a study about the use of parallel processing techniques on the execution of production planning algorithms, targeting both improvement in a manufacturing system as seek improvement on answer's response time and precision when stochastic events that affects the scheduling of a factory's production line occurs. The parallel processing will occur by using of GPUs (Graphics Processing Units) developed with the CUDA architecture, which enables such devices to process data in a parallel way using the graphic processor`s cores ("CUDA cores"). The programming language used on CUDA enabled GPUs is such as the "CUDA C" which it is in fact the traditional "C" language with the addition of massive parallel programming control functions. Also there will be the development of tests which will show the advantages on the GPUs' parallelism at the execution of search algorithms codes (heuristics) in contrast with their sequential version.

Key words: production planning – scheduling of manufacturing systems – CUDA GPUs – parallel processing – CUDA scheduler

SUMÁRIO

1 INTRODUÇÃO 13

1.1 OBJETIVOS 14

1.2 DESCRIÇÃO DO CONTEÚDO DO TRABALHO 14

2 O PROBLEMA DE ESCALONAMENTO EM SISTEMAS DE MANUFATURA 15

2.1 SISTEMAS DE MANUFATURA 15

2.1.1 SISTEMA DE MANUFATURA CELULAR 16

2.1.2 SISTEMAS FLEXÍVEIS DE MANUFATURA 18

2.2 ESCALONAMENTO EM SISTEMAS DE MANUFATURA 20

2.2.1 PROGRAMAÇÃO MATEMÁTICA PARA PROBLEMAS DE ESCALONAMENTO 21

2.3 GRAPHIC PROCESSING UNIT (GPU) 22

2.3.1 PARALELISMO EM PROCESSADORES 23

2.3.2 PARALELISMO EM GPU'S 25

3 SIMULADOR DE PLANEJAMENTO DE PRODUÇÃO COM AUXÍLIO DE GPU'S 28

3.1 ESPECIFICAÇÃO 28

3.2 ESTRUTURA DE UMA FÁBRICA COM CONTROLE AUTOMATIZADO 29

3.3 O ESCALONADOR CUDA 32

3.3.1 PARÂMETROS DE EXECUÇÃO DO ESCALONADOR CUDA 33

3.4 USANDO OS RECURSOS DA GPU 36

3.5 ALGORITMO DE BUSCA DO ESCALONADOR CUDA 37

4 TESTES 45

4.1 INTRODUÇÃO AO CASO DE TESTE 45

4.2 SITUAÇÃO I 48

4.3 SITUAÇÃO II 53

4.4 SITUAÇÃO III 55

4.5 DESEMPENHO 61

5 CONCLUSÕES 65

5.1 DIREÇÕES FUTURAS 65

5.1.1 CONTROLE DE ESTOQUE E SISTEMAS DE MANUSEIO DE MATERIAIS 66

5.1.2 PLANEJAMENTO DO LAYOUT DA FÁBRICA 66

6 REFERÊNCIAS 67

Capítulo 1

Introdução

Ao decorrer da última década, os processadores gráficos (popularmente conhecidos como *"placa de vídeo"*) apresentaram um avanço muito superior às demais áreas da computação. Estes avanços permitiram que as Unidades de Processamento Gráfico (GPUs) fossem usadas para fins muito além dos quais estes dispositivos foram inicialmente projetados.

Os primeiros dispositivos – denominados controladores gráficos – surgiram no início da década de 1980 e exerciam somente a função de desenhar traços e caracteres bitmap nas telas dos computadores industriais da época. Atualmente, as GPUs são complexos circuitos eletrônicos equipados com memória dedicada e processadores potentes capazes de processar frames a uma velocidade impressionante, gerando imagens de alta qualidade com tecnologias avançadas como "pixel shaders" e dinâmica de fluidos.

Além de toda essa capacidade para lidar com imagens, as gerações de GPUs CUDA do fabricante NVIDIA oferecem a linguagem CUDA C aos programadores e desenvolvedores, permitindo que estes usufruam destes dispositivos para processamentos complexos de dados.

A capacidade e complexidade de processamento das GPUs CUDA é aproveitado em diversas áreas além da computação, como é o caso da área biomédica onde muitas pesquisas e estudos são desenvolvidos usando tal tecnologia para o processamento de algoritmos relacionados ao problemas de sequenciamento genético como o de Smith-Waterman por exemplo.

O uso da tecnologia das GPUs CUDA vai muito além da parte teórica, aplicando-se também à parte prática como a análise de imagens geoespaciais e também análises geológicas. As principais áreas de atuação da tecnologia CUDA podem ser encontradas em [1], bem como dados interessantes sobre comparativos de desempenho de processamento entre GPU e CPU.

1.1 Objetivos

Este trabalho teve por objetivo avaliar a aplicação de computação paralela usando GPUs no escalonamento de produção em sistemas de manufatura. Mais especificamente foi avaliado seu uso em escalonamentos por busca heurística, que são problemas inerentemente de complexidade exponencial.

A base do projeto foi o trabalho desenvolvido por Manacero [2], que formulou um sistema híbrido com busca heurística auxiliada por um sistema especialista. No presente trabalho, o sistema especialista foi abandonado, sendo substituído pela paralelização da busca heurística através de GPUs.

1.2 Descrição do conteúdo do trabalho

No capítulo 2, se dará a aquisição de conteúdo e conhecimento sobre os diversos tipos de sistemas de manufatura, planejamento de produção e alguns conceitos básicos sobre processadores gráficos.

O capítulo 3 ficará responsável por introduzir e especificar o simulador proposto neste trabalho, bem como sua relação com o problema de escalonamento em sistemas manufatureiros. Será ainda explicado o contexto do sistema no qual o escalonador se insere.

No capítulo 4 será apresentado em detalhes um caso de teste, o qual passará por três situações nas quais se fará necessário recorrer ao escalonador desenvolvido para que este encontre uma solução para a dada situação. Será mostrado e comentado passo a passo seu funcionamento e os resultados obtidos.

No capítulo 5 dar-se-á um parecer conclusivo sobre o trabalho aqui desenvolvido bem como algumas considerações breves, seguido das referências encontradas ao longo do mesmo.

Capítulo 2

O problema de escalonamento em sistemas de manufatura

2.1 Sistemas de Manufatura

A definição de um sistema de manufatura é tão ampla e complexa quanto a abrangência do termo. Desde uma simples e tradicional fabriqueta de calçados, onde não mais do que dois ou três familiares integram uma linha serial de produção, até uma multinacional fabricante de placas de circuitos impressos (PCB – Printed Circuit Board) com milhares de funcionários e uma complexa rede de máquinas automatizadas e com funções que variam em tempo de execução, tudo pode ser classificado como um sistema de manufatura.

A princípio, se pensarmos em uma fábrica com um número reduzido de funcionários e somente umas poucas máquinas realizando sempre a mesma função, talvez seja um tanto irrelevante e desnecessário pensar em elaborar um método que otimize sua linha de produção e maximize seus lucros uma vez que, sem muito esforço ou sequer conhecimento técnico, o ótimo do problema é obtido de uma maneira analítica simples. Em contrapartida, na medida em que novas máquinas com funções não somente mais complexas, mas também variadas vão sendo adicionadas à linha de produção, a variedade de produtos aumenta e, por consequência, há também um aumento na complexidade do planejamento da linha de produção.

Nas últimas 3 décadas inúmeros estudos, cada vez mais aprofundados, vêm sendo feitos na área de planejamento de produção em sistemas manufatureiros, provando que o sucesso de uma indústria está intimamente relacionado com suas estratégias de produção, especialmente quanto a respeito da organização e disposição do maquinário na planta da fábrica. Para tanto, graças ao intenso esforço acadêmico e científico voltado ao assunto, inúmeras técnicas de planejamento e controle da produção - bem como de controle de recursos e estoque - foram

idealizadas, concretizadas, aplicadas e consolidadas. Dentre essas técnicas, pode-se citar algumas principais tais como:

a) Material Handling System (MHS): Elaboração do traçado (layout) da linha de produção onde há uma preocupação com a eficiência do translado do material entre as diversas fases da produção. Devido ao alto nível de complexidade no desenvolvimento de modelos para estes sistemas, faz-se grande uso de eventos discretos de simulação (DES – Discrete Event Simulation) com o auxílio da computação.

b) Cellular Manufacturing System (CMS): Sistema de manufatura baseado em células. Trata-se de um sistema onde peças com características de produção semelhantes são agrupadas em uma família para serem produzidas por uma célula de produção, que se trata de um conjunto de máquinas especializadas designadas a produzir aquela determinada família de peças.

c) Flexible Manufacturing System (FMS): Um sistema de manufatura flexível possui a habilidade de resposta em tempo real a possíveis variações na demanda por produtos na linha de produção. Outra característica deste tipo de sistema é a capacidade de adaptação após a ocorrência de eventos estocásticos, tais como falhas de máquinas da linha de produção e variações nos tempos de processamento.

Outras inúmeras técnicas para organização da linha de produção estão presentes na literatura. Negahban e Smith em [3] indicam vários estudos tanto sobre os tópicos mencionados acima quanto sobre muitos outros existentes na literatura, os quais não serão vistos neste trabalho por fugirem do escopo do mesmo.

A partir deste ponto, uma breve comparação será feita entre os sistemas de manufatura celular e os sistemas de manufatura flexíveis, a fim de demonstrar e justificar a escolha dos segundo como foco para o sistema a ser especificado e elaborado mais adiante no capítulo 3 deste trabalho.

2.1.1 Sistema de Manufatura Celular

Dentre as principais classificações de organização de sistemas de manufatura, encontramos os *sistemas de manufatura celular* (CMS). Estes sistemas possuem características peculiares onde peças com especificações de produção semelhantes são agrupadas, gerando o que, na

literatura, se chama de ***"família"***. Uma família de peças é processada por uma ***"célula"***, i.e. um conjunto de máquinas cujas funções são semelhantes e compatíveis com as necessidades de produção da família atendida. A figura 2.1 exemplifica um CMS de uma fábrica de pregos e parafusos com duas células e que produz quatro tipos de peças (dois tipos de parafusos, um prego e uma porca), as quais integram duas famílias distintas.

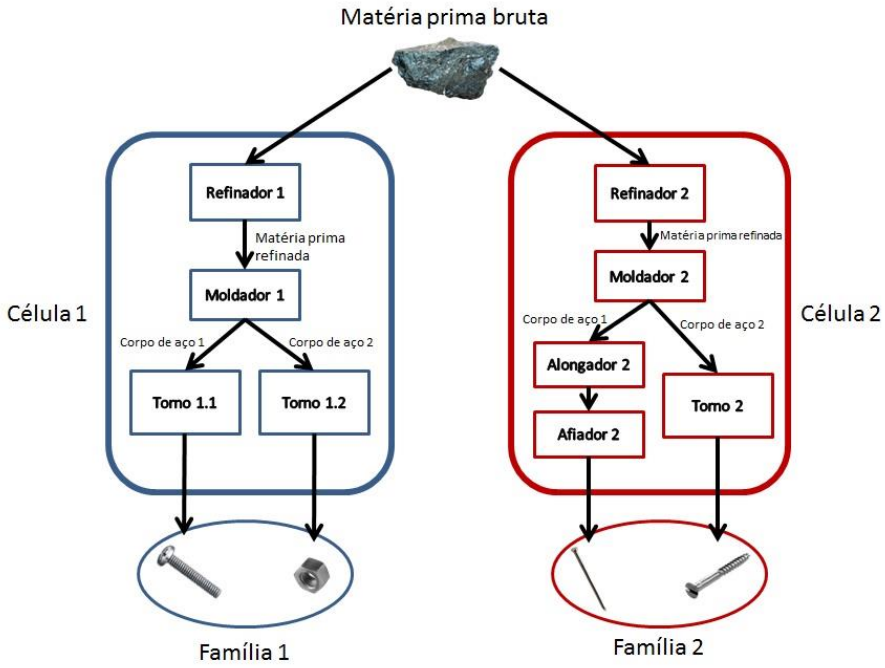

Figura 2-1. Exemplo de uma fábrica com um CMS.

Fica visível que a estrutura da linha de produção de um CMS é muito bem organizada e especializada, fatos que atribuem a este tipo de sistema de manufatura tempos de produção mais rápidos, maior facilidade no manuseio de materiais e no planejamento da linha de produção, maior domínio no controle de custos etc. Entretanto, tantos benefícios são contrabalanceados por uma grande desvantagem: a criação de partições de produtos (famílias).

A rigidez na disposição dos componentes (máquinas) da linha de produção de um CMS é um fator agravante que torna este tipo de sistema altamente sensível a qualquer tipo de falha de máquina, variações nos recursos e até mesmo eventos estocásticos. Ainda na figura 2-1, podemos observar que uma falha em qualquer componente da linha de produção levaria à suspensão do fornecimento de ao menos um tipo de produto, uma vez que a especialidade de cada máquina impede uma realocação das ligações entre elas.

Uma boa fundamentação sobre CMS pode ser encontrada em [4], onde Pitombeira Neto e Gonçalves Filho aprofundam o assunto e discutem os diversos problemas destes sistemas para, em seguida, apresentarem um modelo simulação fazendo uso de um algoritmo genético para busca da solução ótima.

2.1.2 Sistemas Flexíveis de Manufatura

Em contraste à abordagem rígida de estruturação da planta de fábrica em um CMS, existem os *sistemas flexíveis de manufatura* (FMS), os quais são capazes de contornar eficientemente os mesmos problemas que, para os CMS, são de difícil tratamento.

Um FMS se traduz em uma linha de produção integrada, onde as máquinas são multifuncionais, i.e. possuem a capacidade de realizar funções variadas. Estas máquinas são numericamente controladas (NC) por um sistema de controle computadorizado, sendo que tal sistema é também responsável por outros controles como, por exemplo, o do *sistema automatizado de manuseio de materiais* (AMHS) presentes na estrutura de um FMS.

Eventos como variações nos níveis de demanda dos produtos em produção, problemas com estoque de materiais e recursos e eventos estocásticos relacionados às máquinas (quebra, variação de tempo de produção etc.), são tratados em tempo real pelo sistema computacional ligado à linha de produção. Este sistema é responsável por estabelecer as funções de cada máquina e também o relacionamento dentre as mesmas em resposta aos possíveis eventos citados anteriormente. A figura 2-2 mostra um fluxograma tópico sobre o funcionamento do controle computacional de uma planta de fábrica que segue as características de um FMS. Sempre que há uma falha ou evento que comprometa o funcionamento da planta de produção, o sistema de controle computacional é avisado. Este por sua vez, busca a melhor solução para o evento ocorrido e responde reprogramando as funções e relações de máquinas e recursos.

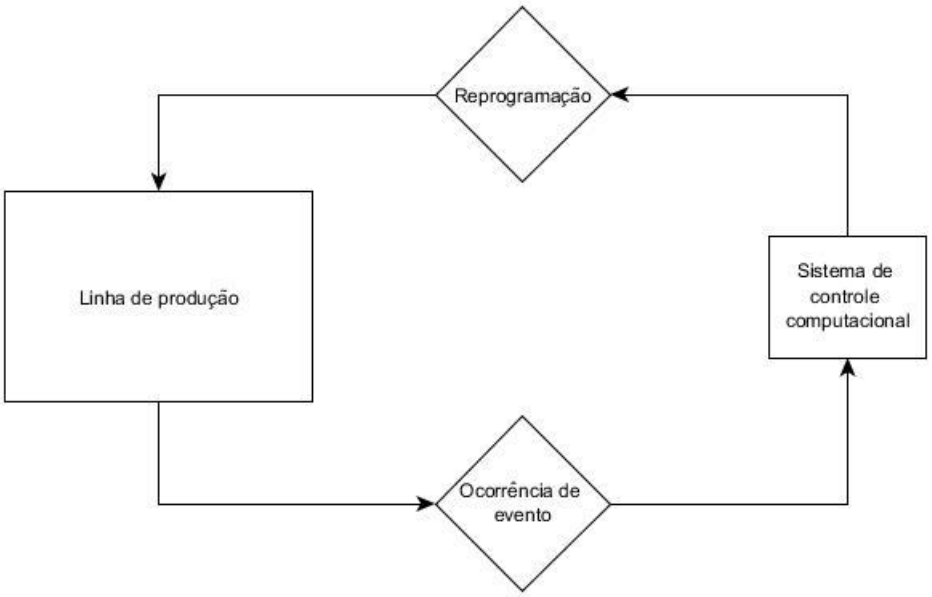

Figura 2-2. Fluxograma simples de um FMS

Nas últimas três décadas, uma grande atenção tem sido dada aos FMS especialmente quando da investigação sobre o uso de técnicas de escalonamento em sistemas flexíveis. Uma boa referência sobre o assunto pode ser encontrada em [5] e [6], onde uma análise mais detalhada é realizada quando dos escalonamentos em sistemas flexíveis de manufatura.

Apesar da capacidade de adaptação obtida pela flexibilidade e dos benefícios proporcionados, os FMS não estão livres de problemas. A principal preocupação em relação aos sistemas flexíveis está relacionada ao planejamento, escalonamento e controle do sistema de produção. O ponto em comum entre tais problemas é que todos devem ser tratados pelo sistema computacional responsável pelas tomadas de decisões de planejamento da linha de produção, fazendo deste o principal motivo para o desenvolvimento do modelo computacional proposto no capítulo 3 deste trabalho.

2.2 Escalonamento em Sistemas de Manufatura

O escalonamento em sistemas de manufatura é, sem dúvida, a parte fundamental para a maximização dos lucros e a minimização dos prejuízos no planejamento de produção de uma fábrica. Pinedo em [7] define o escalonamento em sistemas de manufatura como sendo um processo de tomada de decisão cujo objetivo é otimizar a alocação de recursos para as tarefas ao longo do período de produção.

Os termos genéricos “tarefas” e “recursos” podem se traduzir de maneira diferente em cada tipo de sistema. Em uma fábrica de vestimentas, as tarefas podem ser todos os passos necessários (corte, costura, tingimento etc.) durante o processo de produção, enquanto que os recursos seriam as máquinas ou funcionários humanos responsáveis por cada uma dessas funções. Os sistemas de manufatura não se limitam a somente fábricas e indústrias, tão pouco o fazem as técnicas de escalonamento. Como dito anteriormente, se imaginarmos uma unidade de uma rede de “fast-food”, as tarefas poderiam ser traduzidas como os passos de preparação dos ingredientes e montagem dos produtos alimentícios, bem como os recursos seriam tanto os ingredientes ainda não preparados quanto os funcionários e máquinas presentes durante o processo de preparo.

Em suma, o escalonamento de processos em uma fábrica diz respeito à possibilidade de atender ou não um pedido dentro do prazo estipulado e, para tanto, inúmeras variáveis devem ser levadas em consideração. Usar exemplos é sempre uma boa forma de entendimento e esclarecimento. Assim, Pinedo [7] traz o exemplo de uma fábrica de embalagens, o qual será demonstrado a seguir.

Fábrica de embalagens: Em uma fábrica que produz embalagens para cimento, carvão, rações etc., a matéria prima básica são rolos de material sintético. A linha de produção é composta por três passos: impressão do logo, colagem das laterais da embalagem e selamento da boca. Se imaginarmos que em cada fase da produção o número de máquinas pode variar e que cada uma dessas máquinas, por não serem idênticas, apresentam variações no tempo de realização de uma dada função, já teremos uma boa complexidade de adversidades. Não bastando, temos ainda que cada ordem possui um número específico de um certo tipo de sacola e um determinado prazo de entrega. Sabe-se também que o tempo de produção de um pedido varia de acordo com o tamanho do lote (número de peças).

Atrasos podem e vão acontecer! No exemplo acima o atraso no prazo de entrega de um determinado pedido resulta em penalidades contratuais, as quais variam seus valores de acordo com a importância e magnitude do pedido e do cliente. Desta forma, uma das missões mais importante do escalonamento é minimizar este tipo de perda. De forma análoga, um sistema manufatureiro bem estruturado e com uma política de escalonamento bem fundamentada reagindo aos acasos em que o sistema está sujeito de forma a atender a maior parte dos prazos com a maior eficiência no uso de recursos, terá o lucro maximizado.

Não há dúvidas que o problema do escalonamento em sistemas de manufatura é um assunto que merece (e de fato recebe) grande atenção quanto a estudos e desenvolvimento de novos recursos e tecnologias, tal como será proposto neste trabalho mais adiante.

2.2.1 Programação matemática para problemas de escalonamento

Embora não esteja diretamente ligada a este trabalho, a programação matemática deve ser mencionada uma vez que está intimamente ligada ao problema de escalonamento.

Analisando um sistema de manufatura, não da forma como fora feita na sessão anterior, mas a partir de uma visão matemática, podemos conceituar o problema de escalonamento com a seguinte equação básica de metadados:

Maximizar A utilização dos recursos e o throughput

s.a.

- Restrições de interdependência entre processos
- Restrições de precedência entre tarefas
- Disponibilidade de recursos

Os problemas de escalonamento, quando analisados matematicamente, em sua grande maioria resultam em problemas não polinomiais (NP) e que assim acabam por não gerar uma boa aproximação do ótimo. Tal complexidade se deve à infinidade de variáveis presentes nos problemas de escalonamento, tornando-se inviável e muitas vezes impossível (quando tais variáveis forem de natureza probabilística) adicionar todas as restrições aos problemas de otimização. Com tal relaxação das restrições, as aproximações encontradas não são satisfatórias por estarem muito longe do ótimo. Blazewicz, Dror e Weglarz [8] fazem uma análise aprofundada sobre diversos problemas de programação matemática nos diversos tipos de linhas de montagem.

Visto que diversas relaxações devem ser feitas para tornar factível o problema de otimização mencionado acima, é neste ponto que se introduz o uso da computação para simulações de problemas complexos. Uma simulação por computação pode gerar resultados muito mais próximos do ótimo por ser capaz de agregar ao problema simulado todas - ou grande parte delas - as variáveis antes ignoradas.

2.3 Graphic Processing Unit (GPU)

Os conceitos apresentados até agora neste trabalho têm o objetivo de introduzir o problema para o qual a solução será proposta no capítulo seguinte. Entretanto, é necessário a introdução de mais alguns conceitos, agora, relacionados à GPU.

As unidades de processamento gráfico (GPU’s) são normalmente utilizadas para a exibição de imagens nos dispositivos de saída (monitores) de um computador. Para tanto, as GPU’s contam com um processador sofisticado que é capaz de trabalhar com um alto nível de paralelismo. A seguir, serão apresentados alguns tópicos relacionados ao assunto e que ajudarão a compreender o funcionamento das GPU’s.

2.3.1 Paralelismo em processadores

Quando mencionamos *"paralelismo"* em um processador, na verdade estamos nos referindo à capacidade deste executar diversos processos simultaneamente (O termo *"simultaneamente"* é usado de maneira errônea neste caso, pois na verdade o que acontece é, quem diria, o escalonamento destes processos). Um processo em execução no processador nada mais é do que um trecho de código a ser computado pelo processador. Cada processo pode ser considerado uma *"thread"* (linha, em inglês), logo, por implicação, uma thread é uma pilha de código e cada thread pode conter threads filhas, i.e., processos dependentes do processo mais externo ou ainda, trechos de códigos dentro do código mais externo. Confuso? Sim, é mesmo.

Apesar da complexidade destes conceitos e da dificuldade em entendê-los, são eles os responsáveis pela escolha do computador como uma ferramenta poderosa de simulação. A possibilidade de paralelismo no processamento de informações torna a computação ideal para a simulação de problemas de escalonamento em sistemas de manufatura, onde a complexidade do planejamento de produção pode ser tratada de forma satisfatória usando as vantagens oferecidas pelo processamento paralelo de informações.

Para ressaltar as vantagens mencionadas, vamos exemplificar o tratamento de um problema de escalonamento por um processador serializado, i.e. um processador "cego" que não consegue lidar com mais do que um processo por vez e comparar o tratamento do mesmo problema por um processador que suporte o paralelismo. A figura 2-3 mostra um grafo dirigido que representa um problema de escalonamento de produção, onde os vértices representam os estágios da produção de uma peça e as arestas representam os caminhos possíveis até o estágio final de produção, o custo em cada aresta representa o montante de recursos necessário para a produção da peça no estágio seguinte caso aquele caminho seja escalonado.

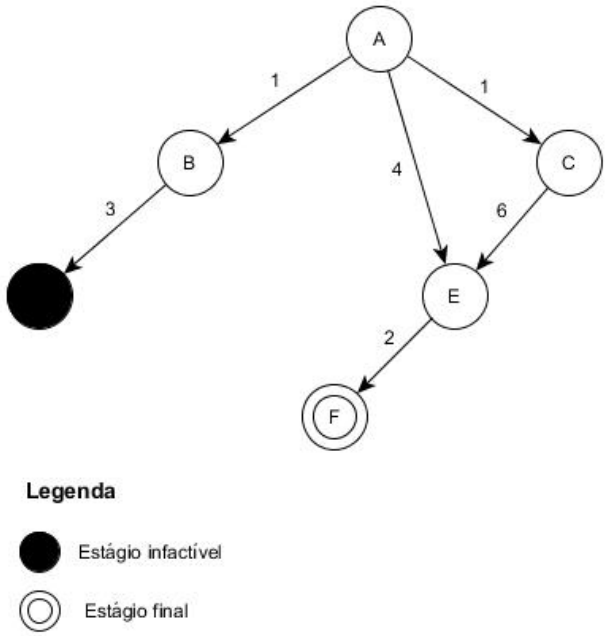

Figura 2-3. Estágios de produção de uma peça.

Suponha que um processador serial seja responsável por tratar este problema e que o algoritmo usado na simulação para a busca do caminho ótimo seja o algoritmo do menor caminho. Supondo também que exista um sistema externo de controle de escalonamento da linha de produção e que esteja esperando a resposta do simulador para que este (o sistema escalonador externo) possa estabelecer o melhor caminho para a produção da peça em questão, o tempo de espera pela resposta será equivalente ao tempo de processamento das três rotas possíveis entre o estágio A e o estágio F. Este problema acontece por haver uma única thread rodando, ou seja, a thread principal responsável pelo único processo residente no processador serial.

Supondo agora que o mesmo problema seja tratado por um processador habilitado para o paralelismo como é mostrado na figura 2-4. Quatro threads são disparadas, onde uma será a thread principal e que controlará as outras três threads filhas sendo estas responsáveis cada uma por calcular o custo de um dos três caminhos possíveis. No final do processamento de cada thread filha, a mesma irá retornar à thread principal o resultado do custo do respectivo caminho. A thread principal por sua vez retornará ao escalonador o caminho com o menor custo.

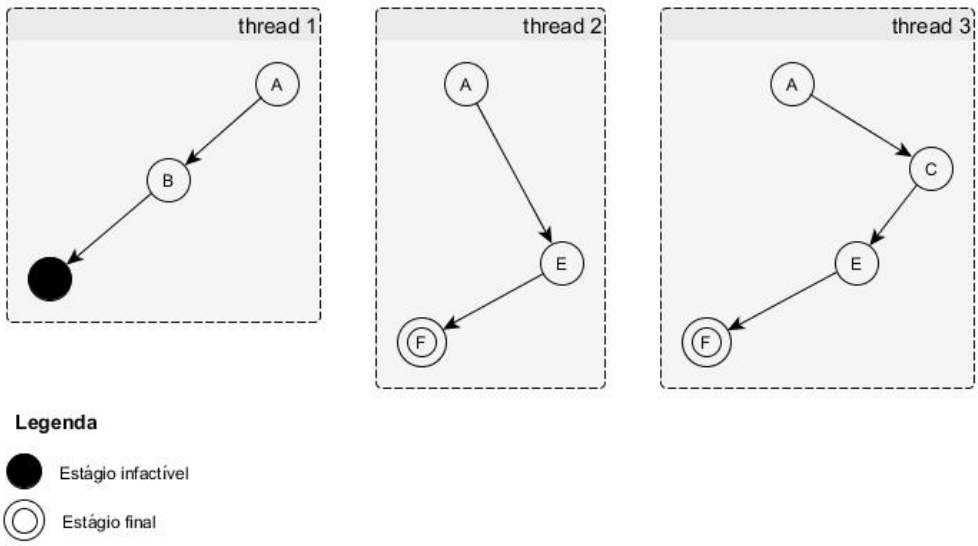

Figura 2-4. Estágios de produção de uma peça com o uso de threads.

Com o uso de threads retratado na figura 2-4, o tempo de espera pela resposta será equivalente a 1/3 do tempo oferecido pelo processador serial. Toda esta demonstração prova que a computação paralela é uma ferramenta realmente poderosa e que, em escala de uma indústria real, o uso desse paralelismo é fundamental.

2.3.2 Paralelismo em GPU's

A definição e exemplificação de paralelismo visto até agora servirá para demonstrar o quão poderoso é este recurso em uma GPU. Primeiramente, devemos especificar que os processadores gráficos dos quais trataremos de agora em diante são aqueles produzidos pelo fabricante *"NVIDIA"* e que são construídos com a arquitetura *"CUDA"*, necessária para o processamento de códigos estruturados na linguagem imperativa *"CUDA C"* que foi desenvolvida pelo mesmo fabricante em questão.

A linguagem CUDA C é uma modificação da linguagem C, a qual é bem conhecida por implementar os sistemas operacionais derivados do *"UNIX"* e também por ser uma ferramenta útil e muito poderosa na programação com manuseio direto de endereçamento de memória (ponteiros). As modificações presentes na linguagem CUDA C se tratam, em sua

maioria, de funções projetadas para o controle da comunicação, endereçamento e transferência de dados entre a memória principal do computador (*"host"*) e a do dispositivo gráfico (*"device"*) que contenha o processador gráfico construído sobre a arquitetura CUDA. Uma introdução bem didática sobre a arquitetura CUDA e a linguagem CUDA C é apresentada em [9].

Com base nessas mudanças na estrutura da linguagem C, ganhamos o poder de usar o elevado nível de paralelismo presente nas GPU's com arquitetura CUDA. Tal capacidade de processamento paralelo é alcançada com a criação de até cinco dimensões de threads!

Uma GPU habilitada para a arquitetura CUDA possui uma divisão de memória um tanto peculiar que possibilita ao programador planejar exatamente como ele a irá alocar, mais especificamente com duas dimensões de *"blocos"* e três dimensões de threads como é demonstrado na figura 2-5.

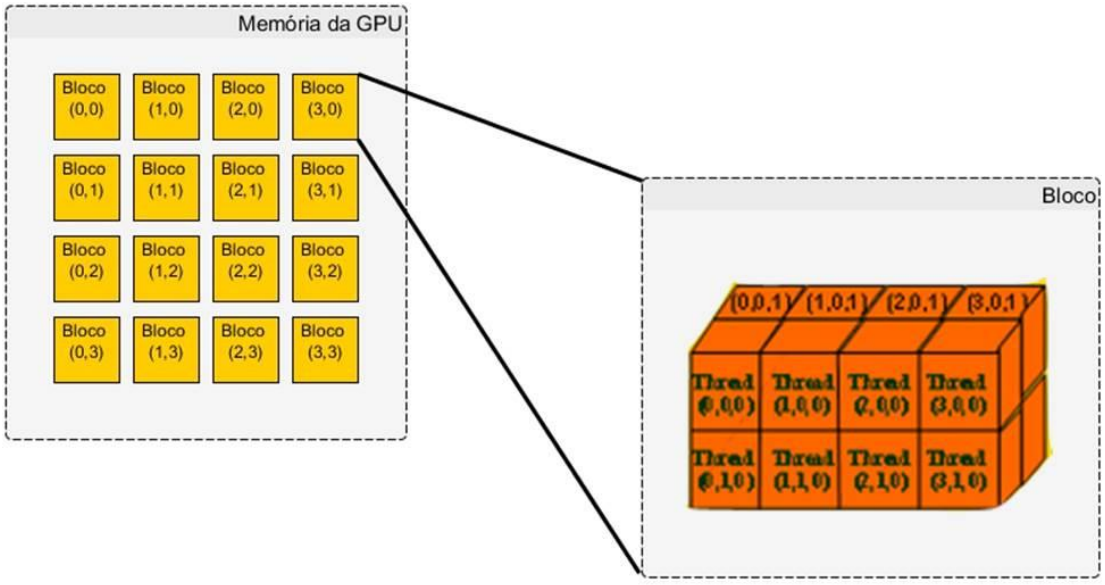

Figura 2-5. Blocos e threads na memória de uma GPU.

Do mesmo modo como foi apresentado anteriormente na figura 2-4, aqui também podemos inserir um trecho de código em cada thread rodando dentro de cada bloco, o que nos dá a habilidade de executar códigos com um paralelismo de altíssimo nível e grande complexidade. O resultado do uso desta capacidade de paralelismo é tal que podemos executar simulações pesadas e obter resultados em um tempo relativamente curto quando comparado com o tempo que um processador comum levaria para executar a mesma tarefa, ainda que este seja capaz de paraleliza-la. O motivo é tal que os processadores comuns, apesar

de possuírem capacidade de paralelismo, não têm um estrutura de memória dedicada a este tipo de função assim como são as GPU’s CUDA.

Capítulo 3

Simulador de planejamento de produção com auxílio de GPU's

3.1 Especificação

Neste capítulo se dará o desenvolvimento de um sistema simulador de planejamento de produção o qual terá seu código implementado na linguagem CUDA C para ser executado sobre GPU's construídas com a arquitetura CUDA.

O trabalho aqui desenvolvido tem por objetivo projetar o *"kernel"* (núcleo) de um sistema maior e mais complexo responsável pelo controle de uma fábrica que adota a política de manufatura flexível para estruturação e planejamento da produção. Como já visto anteriormente o funcionamento de um FMS, sabe-se que este tem a característica de flexibilidade quando da interligação entre as máquinas e das funções por elas realizadas.

As próximas sessões demostrarão as atribuições e restrições do funcionamento do modelo proposto.

3.2 Estrutura de uma fábrica com controle automatizado

A estrutura geral do funcionamento da fábrica fictícia, onde o modelo proposto será executado, é mostrado na figura 3-1. Como dito, trata-se de um modelo fictício e, portanto, muitas das suposições e atribuições sobre este podem não corresponder a um caso real.

Como o foco deste trabalho está voltado somente para o desenvolvimento do módulo "Simulador de Planejamento de Produção CUDA", apenas alguns detalhes de funcionamento dos outros módulos serão abordados, não mais do que o suficiente para a compreensão do módulo mencionado como foco. Após a figura 3-1 se encontra uma breve descrição dos diversos componentes que estruturam o funcionamento do modelo de controle fictício.

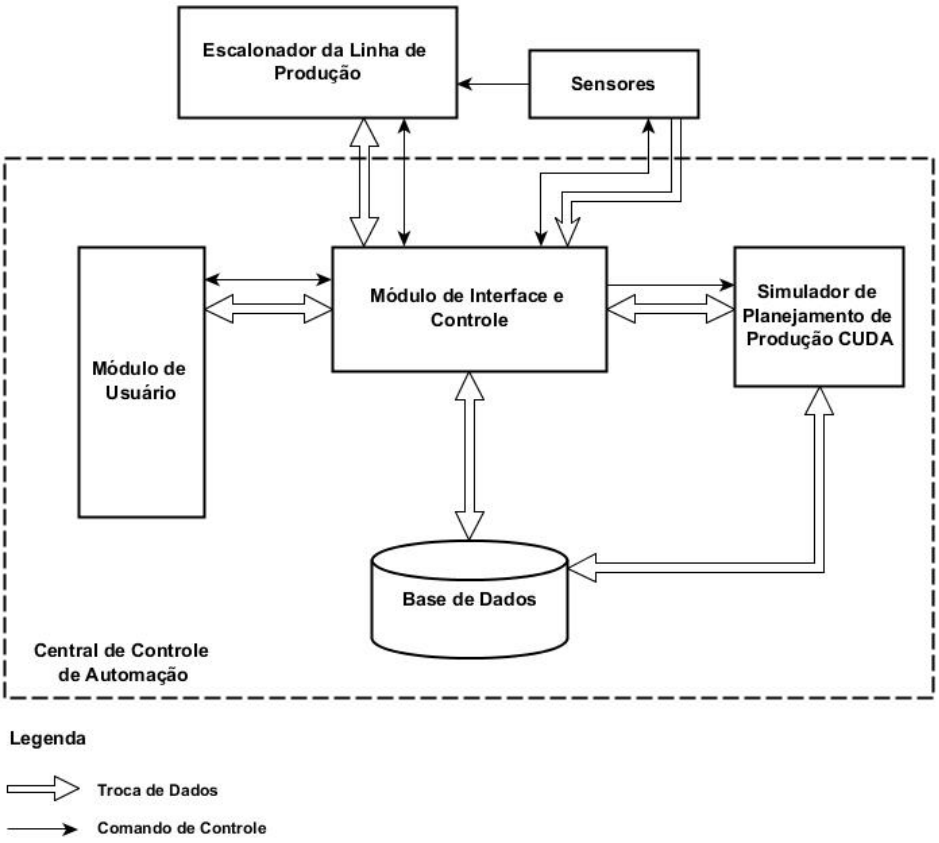

Figura 3-1. O modelo de controle de produção automatizado proposto.

Como fica evidente pela figura 3-1, o simulador de planejamento de produção CUDA é somente uma pequena, porém fundamental, parte de um sistema muito maior e mais

complexo de controle automatizado. A funcionalidade básica de cada módulo se dá da seguinte maneira:

a) Escalonador da Linha de Produção

Este módulo representa o controle direto da linha de produção, i.e. as máquinas, as ligações entre estas e o controle de outros recursos. O escalonador da linha de produção recebe do módulo de interface e controle ordens diretas sobre como deve ser feito o escalonamento da linha de produção. Exemplificando, quando uma nova peça passa a ser produzida ou um novo pedido com maior prioridade (seja por um prazo apertado ou devido a um cliente muito exigente) do que o atual é recebido, o escalonamento da linha de produção deve ser alterado.

Para que uma alteração de escalonamento aconteça, o módulo de interface deve ficar ciente dos novos planos de escalonamento e em seguida enviar uma mensagem de controle ao módulo escalonador da linha de produção juntamente com os dados do novo escalonamento. O módulo escalonador interrompe o funcionamento atual e realiza o replanejamento da linha de produção de acordo com o escalonamento proposto.

b) Sensores

Os sensores são sistemas conscientes que estão espalhados pela fábrica e ligados à linha de produção, muitas vezes se encontram até mesmo dentro ou conectados às máquinas. Os sensores enviam sinais periodicamente ao módulo de interface e controle de forma a manter atualizados os dados da linha de produção como um todo. Estes sinais podem conter informações sobre o estado de cada máquina (temperatura, velocidade, desgaste etc.), sobre a quantidade e disponibilidade de recursos e sobre muitos outros tipos de informação que variam de caso para caso.

Os sensores são também responsáveis por emitir sinais que param completamente a produção caso ocorra alguma falha crítica como, por exemplo, um incêndio ou um acidente com algum funcionário. Estes sinais são tratados com prioridade pelo escalonador da linha de produção fazendo com que todo o sistema entre em estado de parada total. O módulo de interface e controle é avisado posteriormente fazendo com que este aguarde uma instrução do módulo de usuário sobre como reagir.

c) Módulo de Interface e Controle

Este é o módulo central do sistema proposto. Sua função é tratar todas as transações entre os diversos módulos do sistema fazendo com que estes sejam capazes de se comunicarem. Outra

função de grande importância é sua ligação com o banco de dados, o qual contém informações importantes sobre o estado do sistema e que são necessárias para o funcionamento dos demais módulos e do sistema como um todo.

O módulo de interface e controle faz o intermédio de comunicação com o banco de dados sempre que qualquer outro módulo faz uma requisição de dados. Além disso, as tomadas de decisões automáticas baseadas em regras de reação a eventos são também uma importante, se não a mais, função de controle.

Quando uma troca de escalonamento é necessária, primeiramente o módulo de interface e controle busca na base de dados uma regra conhecida para resolver o problema, buscando também se há uma situação conhecida pela qual já fora apresentado um escalonamento factível em algum momento naquele sistema de produção. Quando nada é encontrado (nem regra, nem casos passados), o módulo de interface e controle aciona o módulo simulador de planejamento de produção CUDA o qual irá computar uma solução para o problema através do Escalonador CUDA, foco principal deste trabalho e que será detalhado mais adiante.

d) Módulo de Usuário

O módulo de usuário nada mais é do que uma interface amigável ao usuário, onde dados são inseridos e obtidos em formatos familiares ao ser humano. É no módulo de usuário onde ocorrem as tomadas de decisões críticas como as mencionadas anteriormente durante a descrição do módulo de sensores. Quando o sistema se encontra em tal situação, é necessário a intervenção do engenheiro ou técnico responsável, a qual se dá através da inserção de dados pelo módulo de usuário. Outros casos específicos como uma parada não programada da linha de montagem também são feitos através do módulo em questão.

e) Simulador de Planejamento de Produção CUDA

Responsável por abrigar o cerne deste trabalho, o módulo simulador de planejamento de produção CUDA é o *"lado esquerdo do cérebro"* (no ser humano, responsável pela lógica e raciocínio exato) do sistema proposto. É nele onde acontece toda a computação pesada da solução de problemas de escalonamento.

Este módulo conta com a presença de um dispositivo gráfico potente habilitado para a execução de códigos com paralelismo maciço, fazendo uso da linguagem CUDA C e da arquitetura CUDA presente em tal dispositivo.

Sem mais, a seção seguinte irá tratar em detalhes a especificação e o funcionamento deste sistema.

3.3 O Escalonador CUDA

Sempre que um escalonamento se torna infactível ou um novo escalonamento precisa ser produzido, o escalonador CUDA é acionado. Se em algum momento, uma nova peça é adicionada ao conjunto de peças produzidas pelo sistema manufatureiro proposto e um novo pedido de produção desta peça é estabelecido, indubitavelmente o escalonamento de produção atual deverá ser reestruturado. Tão como, se uma máquina sofre um evento de quebra ou alguma matéria prima entra em estado de falta, o escalonamento também deverá ser recalculado.

Outra situação pode ser também uma simulação proposta pelo usuário, i.e. o engenheiro/técnico responsável pode estar estudando uma reestruturação do sistema e, para concluir e solidificar seus estudos, uma simulação de um novo escalonamento talvez seja a melhor ferramenta. Para tanto, o usuário deverá ter em mãos os dados do sistema estudado e assim inseri-los em um arquivo de configuração, o qual será posteriormente passado ao escalonador. O módulo de interface e controle mais uma vez intermediará a comunicação traduzindo todas as informações necessárias à execução do escalonador - em termos computacionais, irá compilar o código com os dados inseridos e assim gerar um novo arquivo binário para ser executado no escalonador.

É nos casos mencionados acima, bem como em muitos outros, que o escalonador CUDA deve entrar em ação. Primeiramente, o módulo de interface e controle deve ser informado do evento que possivelmente levará a um novo escalonamento para, em seguida, começar uma busca na base de dados sobre possíveis soluções para o mesmo. Quando nenhuma solução for encontrada, o módulo de interface e controle irá então recorrer ao escalonador CUDA.

Para que a ativação do escalonador seja possível, o módulo de interface deverá obter – ou da base de dados, ou dos sensores espalhados pela fábrica – as informações mais atualizadas sobre o estado geral do sistema. Estas informações serão então inseridas em um arquivo de

configuração que será usado para compilar e gerar um arquivo binário o qual, por sua vez, será executado pelo escalonador CUDA.

Em resumo, um arquivo binário nada mais é do que uma versão de “zeros e uns” do código de alto nível escrito pelo programador. Esta versão binária é gerada pelo compilador (“nvcc” no caso da linguagem CUDA C nos sistemas operacionais Linux/Unix) para ser executada diretamente a nível de hardware. Vamos, em seguida, analisar os dados necessários para gerar tal arquivo binário.

3.3.1 Parâmetros de execução do escalonador CUDA

Como já fora dito, a chamada ao escalonador necessita de alguns dados para que o arquivo binário seja gerado. Dentre estes dados estão diversas informações sobre o sistema tais como mostram as tabela 3-1, 3-2, 3-3, 3-4 e 3-5. Mais uma vez, as informações sobre o sistema devem ser constantemente atualizadas, i.e. os dados sobreo estado do sistema devem ser obtidos no momento estritamente antes da execução do escalonador, pois somente desta forma a factibilidade dos resultados e uma boa aproximação do ótimo do problema de escalonamento poderão ser alcançados.

Tabela 3-1. Dados referentes às máquinas.

	inProd	reset	tier
Máquina 1	-	-	-
Máquina 2	-	-	-
Máquina 3	-	-	-
⋮	⋮	⋮	⋮
Máquina n	-	-	-

Tabela 3-2. Relação de tempo de preparo (Setup) entre as peças na máquina M.

De \ Para	Peça 1	Peça 2	Peça 3	. . .	Peça n
Peça 1	0	-	-	. . .	-
Peça 2	-	0	-	. . .	-
Peça 3	-	-	0	. . .	-
.	.	.	.		.
.	.	.	.	0	.
.	.	.	.		.
Peça n	-	-	-	. . .	0

Tabela 3-3. Tempo de produção de cada peça na máquina M de acordo com seu tier.

	prodTime (tier 1)	prodTime (tier 2)	prodTime (tier 3)	. . .	prodTime (tier n)
Peça 1	-	-	-	. . .	-
Peça 2	-	-	-	. . .	-
Peça 3	-	-	-	. . .	-
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
Peça n	-	-	-	. . .	-

Tabela 3-4. Dados referentes aos níveis de atuação das máquinas (tiers).

	size	tail
tier 1	-	-
tier 2	-	-
tier 3	-	-
. . .	. . .	. . .
tier n	-	-

Tabela 3-5. Dados referentes aos lotes a serem produzidos.

	prodType	prodAmount	deadline
Lote 1	-	-	-
Lote 2	-	-	-
Lote 3	-	-	-
⋮	⋮	⋮	⋮
Lote n	-	-	-

A informação contida nas tabelas acima é usada em um complexo algoritmo executado diretamente no dispositivo gráfico, usando sua memória e seu processador. Vamos analisar em detalhes, ao longo deste e do próximo capítulo, o significado e a aplicação de cada um dos dados apresentados nas tabelas acima bem como uma abstração do algoritmo mencionado.

3.4 Usando os recursos da GPU

Para entendermos o paralelismo presente no algoritmo de escalonamento executado na GPU, é preciso antes um entendimento básico sobre como os recursos (memória e núcleos de processamento) desta são alocados e utilizados.

Quando trabalhamos com execução de códigos em GPUs CUDA, estamos fazendo uso de um arsenal pesado de computação paralela. As GPUs com arquitetura CUDA oferecem uma estrutura para a execução simultânea de códigos paralelos, onde há uma combinação de *"blocks"* que podem conter diversas *"threads"* como já fora visto no capitulo anterior. Vamos – de maneira superficial – entender um pouco sobre o conceito destas estruturas.

a) blocks (blocos)

Os blocos são os componentes mais externos da estrutura de memória de uma GPU arquitetada em CUDA. Podendo existir até três dimensões fictícias de blocos, são dentro destes que residem as threads e, por sua vez os códigos a serem executados. O conjunto de todos os blocos forma uma *"grid"*, que nada mais é do que uma forma de se trabalhar com os índices dos blocos. A figura 2-5 oferece um entendimento visual.

b) threads (linhas)

Estas são as unidades fundamentais e que realmente interessam. São as threads as estruturas responsáveis por carregar o código a ser executado na GPU, de forma que em cada bloco podemos ter até três dimensões de threads rodando. Se repararmos, é uma estrutura aninhada em que podemos, nos hardwares mais modernos e potentes, ter até seis dimensões de códigos em execução paralela, muito embora isso dificilmente seja alcançado devido tanto à complexidade de controle de tal estrutura quanto à falta de ocasiões normais para a qual este artifício se faça necessário.

Voltando o foco para o escalonador, as threads são os elementos responsáveis por executar uma cópia do código onde queremos exercer o paralelismo. Isto é feito através da elaboração de um trecho de código genérico, comum a todas as threads em execução. Neste trecho de código, cada thread será identificada pela sua posição dentro do bloco em que residem. Por exemplo, uma aplicação onde 10 blocos são alocados em uma única dimensão e, cada bloco, possui 20 threads também em uma única dimensão, a referência à thread 12 do bloco 3 seria a seguinte (lembrando que os índices começam a partir do zero):

```
if (blockIdx.x ==  2) {
        if (threadIdx.x == 11) {
                foo();
        }
```

Desta forma conseguimos montar uma estrutura complexa onde identificamos cada thread unicamente em tempo de execução, de forma que esta faça somente a tarefa para a qual foi designada, i.e. sem entrar na área de atuação de outras threads. Para não entrar em maiores detalhes sobre a linguagem CUDA C, o livro "CUDA BY EXAMPLE: An Introduction to General-Purpose GPU Programming" [9] foi usado como base fundamental para o desenvolvimento deste trabalho e é uma excelente fonte para aprendizado da linguagem CUDA C, bem como dos recursos da programação em GPUs CUDA.

3.5 Algoritmo de busca do escalonador CUDA

Chegando ao ponto mais interno da elaboração deste trabalho, o algoritmo de escalonamento proposto é executado com base nas informações fornecidas nas tabelas 3-1, 3-2, 3-3 e 3-4.

Primeiramente, de maneira muito simples, aloca-se um bloco na memória da GPU para cada lote a ser produzido no sistema manufatureiro. Em seguida, deve-se calcular a quantidade de threads que deverão ser alocadas em cada bloco. Para isso, deve-se saber quantos *"tiers"* de máquinas existem e quantas máquinas cada um possui.

i. Tiers (Camadas)

Como já fora dito anteriormente, o sistema de manufatura aqui adotado é de caráter flexível e, portanto, considera-se que todas as máquinas possuem funções semelhantes e podem produzir

todas as peças na linha de produção. No entanto, estes sistemas também possuem algumas regras como é o caso dos tiers.

As máquinas são agrupadas em tiers de forma que cada tier produza um estágio da peça como na figura 3-2. Cada máquina pertence a um tier e cada tier tem um número de máquinas, estes dados são designados no arquivo de configuração

Diferentemente de um CMS, tiers NÃO são células. Em um FMS, uma mesma máquina pode integrar vários tiers bastando mudar tal configuração passada nos parâmetros de execução do escalonamento, algo que não é possível em um CMS, pois as máquinas de uma célula são específicas para a função exercida nesta referida célula.

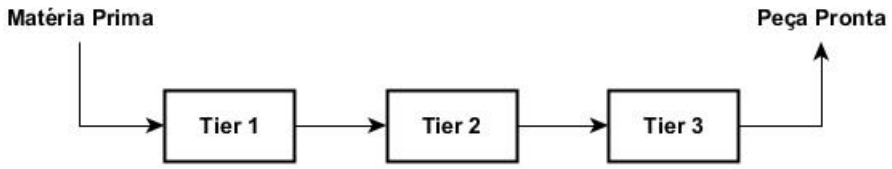

Figura 3-2. Caminho de uma peça pelos tiers de máquinas em um FMS.

De volta ao cálculo do número de threads por bloco, tendo em mãos a quantidade de tiers e o número de máquinas em cada um, podemos calcular o número de threads por blocos pela seguinte equação 3-1:

$$threads_{per_{block}} = tier[1]_{nMachines} * tier[2]_{nMachines} * \dots * tier[n]_{nMachines}$$

Equação 3-1. Obtenção do número de threads por bloco.

A equação acima é na verdade uma análise combinatória onde todas as máquinas devem se relacionar formando um caminho do primeiro ao último tier, com a restrição de que nenhum caminho passe por mais de uma máquina por tier (i.e. máquinas não mandam peças para máquinas do mesmo tier) e também que o tier da máquina seguinte deve sempre ser maior –

em uma unidade – do que o tier da máquina atual. A figura 3-3, por meio de um grafo dirigido, oferece um entendimento adequado ao que está sendo explicado.

Agora que já é conhecido o número de threads por bloco supondo que se conheça o número de blocos, chega a hora de entender como se dá o procedimento de cada thread durante a busca dos caminhos. A partir deste ponto, também será usada as demais informações passadas como parâmetro no arquivo de configuração usado para compilar e gerar o arquivo binário de execução.

Seguindo, faz-se agora uma chamada à função que irá controlar a execução do código na GPU, passando como parâmetros para esta função o número de blocos e o número de threads por blocos. Esta função irá gerenciar a alocação de memória na GPU e entregar uma cópia de seu código para cada thread executar.

A função referida é comumente denominada *"kernel"* mas para evitar confusões com terminologias já mencionadas até este ponto, iremos denomina-la *"Scheduler"* de agora em diante.

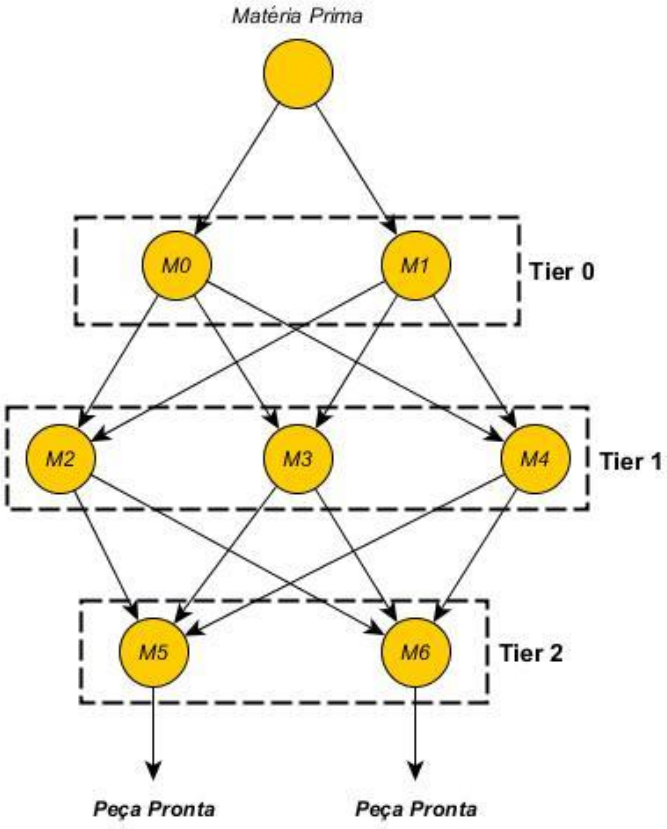

Figura 3-3. Combinação de caminhos.

Baseando-se no exemplo contido na figura 3-3, e no que foi visto até agora, teremos chamado a função *"Scheduler"* da seguinte maneira:

$$Scheduler <<< num_{blocks}, num_{threadsPerBlock} >>> (param_1, param_2, \dots);$$

Onde o num_{blocks} é o número de blocos que equivale ao número lotes a serem produzidos e $num_{threadsPerBlock}$ é o número de threads por bloco calculado a partir da equação 3-1. Os parâmetros entre parênteses são os dados referentes às máquinas, lotes, tiers etc., sendo exatamente os parâmetros inseridos no arquivo de configuração usado para gerar o binário. Estes parâmetros serão usados pelas threads para atingirem seu objetivo principal que é encontrar o caminho com o menor custo dentre todos.

Ao começar sua execução, uma thread irá seguir um caminho pré-determinado por uma heurística definida dentro de seu código. Cada thread irá então buscar um caminho como mostra a figura 3-4.

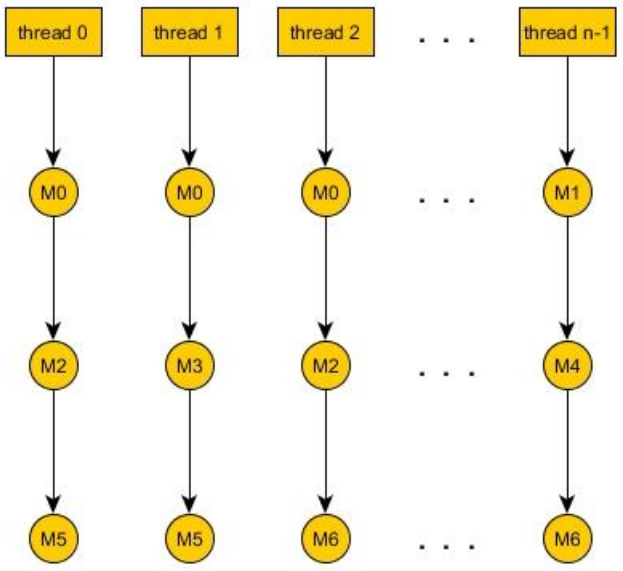

Figura 3-4. Threads executando um caminho único

O que acontece é que, durante a execução de cada thread, a mesma calcula o tempo que o lote referente ao bloco a qual esta thread pertence (i.e. bloco1 = lote1, bloco2 = lote2 etc.) levará para ser produzido através daquele caminho, levando em consideração fatores como:

- o tempo que cada máquina leva para produzir a peça a ser fabricada naquele lote;
- o tempo de setup (preparo) de cada máquina caso esta estivesse produzindo uma peça diferente no escalonamento anterior;
- o tempo de reset (rearme) que cada máquina leva para produzir uma determinada peça;
- a quantidade de peças a serem produzidas no lote em questão.

Para um melhor entendimento da execução dos códigos em cada thread, se faz necessário o entendimento dos dados exibidos nas tabelas 3-1, 3-2, 3-3, 3-4 e 3-5 que ainda não foram mencionados até este ponto.

ii. Setup (preparo)

É o tempo que uma máquina leva para se preparar para produzir uma peça diferente. Em outras palavras é o tempo que cada máquina leva para mudar seu estado. Este tempo depende da máquina e, principalmente, de qual peça esta máquina estava produzindo e qual será a próxima peça a ser produzida. Estes tempos estão armazenados no banco de dados na forma de uma tabela (como mostrado na tabela 3-2) sendo que cada máquina possui as suas próprias informações de Setup, i.e. a sua própria tabela 3-2.

iii. Reset (rearme)

É o tempo que uma máquina leva para se preparar (rearmar) entre o final da produção de uma peça e o início da produção de outra unidade da mesma peça. Esta informação está contida na tabela 3-5, a qual armazena os dados sobre as máquinas da linha de produção.

iv. inProd

Informação sobre a peça que está ou estava em produção naquela máquina. Esta informação é fundamental para o cálculo do tempo de setup das máquinas em cada caminho que cada thread calcula.

v. prodTime

Estes dados se referem ao tempo que cada peça leva para ser produzida em uma determinada máquina M. Cada máquina tem sua própria tabela 3-3, onde se encontra o tempo que a peça

"p" leva para ser produzida na máquina "M" em cada tier "t" ao qual a máquina pertença no atual layout da linha de produção.

vi. size (tamanho) e tail ("cauda")

Na tabela de dados referentes aos tiers (tabela 3-4), estas propriedades remetem à quantidade de máquinas presentes naquele tier e à máquina com o ID (identificador) de tier (difere do ID da máquina) mais baixo dentro daquele tier, respectivamente. Por exemplo, a máquina de ID = 3 pode ser a calda do tier 0 e portanto possuir um tierID = 0. A propriedade tail é usada para efeito de cálculo do trajeto que cada thread deve seguir sem que esta repita um caminho já percorrido por alguma outra thread do mesmo bloco.

vii. prodType, prodAmount e deadline

Estas são as propriedades pertinentes a cada lote a ser escalonado para produção. Respectivamente remetem ao tipo de peça a ser produzida naquele lote (assume-se neste trabalho que um lote contenha somente um tipo de peça), a quantidade desta peça e o prazo de entrega (medido em uma unidade de tempo genérica denominada u.t. – units of time).

De volta ao algoritmo, após todas as threads terem executado seu trecho de código, a função Scheduler retorna ao programa principal um *"buffer"* contendo todos os caminhos possíveis para cada lote naquela configuração da linha de produção, bem como o tempo de "turnaround", i.e. o tempo total de produção do lote para cada caminho. Dentro do tempo de turnaround está incluso o cálculo de todas as propriedades descritas acima.

Com base no tempo de turnaround calcula-se a "folga" que cada caminho gera para a produção do respectivo lote. O cálculo da folga se dá segundo a equação 3-2.

$$looseness_{path} = deadline_{batch} - turnaround_{path}$$

Equação 3-2. Cálculo da folga para produção de um lote.

Estes dados são analisados por uma rotina no código do escalonador onde o melhor caminho de cada lote, i.e. aquele que tem a MAIOR folga, é escolhido para ser o caminho favorito para a produção do respectivo lote.

Dentre os caminhos favoritos de todos os lotes, i.e. aqueles escolhidos anteriormente por apresentarem a maior folga, escolhe-se agora os de MENOR folga para serem produzidos uma vez que estes são os mais prioritários levando em consideração seus prazos de entrega (deadlines).

Os dados são então atualizados, as máquinas escolhidas para a produção dos lotes de maior prioridade são marcadas como “em uso”. O escalonador CUDA é mais uma vez acionado e todo o processo repetido até que seja encontrado o melhor escalonamento para os lotes a serem produzidos, i.e. buscando produzir todos os lotes possíveis dentro do menor tempo possível levando em consideração seus respectivos deadlines.

O algoritmo 3-1 a seguir mostra uma abstração do funcionamento do código do escalonador CUDA, permitindo um melhor entendimento sobre tudo que foi dito sobre o mesmo até este ponto.

```
carregue os dados atualizados do sistema;
carregue os dados sobre os lotes;
calcule a quantidade de caminhos possíveis segundo o layout atual da fábrica;
enquanto houver lotes não escalonados faça
início
        chame a função Scheduler<<num_blocks, num_threads_per_block>>(parameters);
        encontre o caminho de MAIOR folga para cada lote ainda não produzido;
        enquanto houver maquinas livre no tier de menor "size" faça
        início
                encontre o lote de MENOR folga dentre os escolhidos no passo anterior;
                coloque-o em produção;
        fim
        atualize os tempos de cada lote em produção;
        libere as maquinas dos lotes que terminaram sua produção;
        atualize o tempo total decorrido;
fim
retorne o resultado do escalonamento;
```

Algoritmo 3-1. Abstração do funcionamento do escalonador CUDA.

Os resultados gerados pelo escalonador são devidamente tratados pelo módulo de interface e controle o qual, por sua vez, gera instruções e as passa ao módulo escalonador da linha de produção para que este reprograme o escalonamento da linha de produção de acordo com os resultados gerados pelo escalonador CUDA.

Capítulo 4

Testes

Este capítulo será dedicado à demonstração de um caso teste que irá passar por três situações diferentes. Cada uma destas situações irá gerar uma alteração nas configurações do sistema, sendo necessário recorrer ao escalonador CUDA para buscar um novo escalonamento da produção.

Os testes foram realizados sobre a seguinte configuração de hardware:

- Processador Intel Core i7-4700MQ, 2400MHz;
- 16 GB de memória principal DDR3;
- Linux x86_x64;
- CUDA 5;
- NVIDIA GeForce GTX 765M, 2GB GDDR5, 768 CUDA cores;

4.1 Introdução ao caso de teste

Seja uma fábrica que adota o conceito de FMS em sua linha de produção. Esta fábrica, inicialmente, possui sete máquinas que são capazes de produzir três tipos de peças diferentes. O objetivo é fazer uso do escalonador CUDA apresentado no capítulo anterior para tratar da melhor forma as condições que serão impostas ao sistema modelo no decorrer deste capítulo.

As tabelas 4-1 a 4-10 fornecem todos os dados que serão usados pelo escalonador no decorrer dos testes para que este gere o melhor escalonamento em cada situação a ser proposta. Outros dados e alterações da estrutura do sistema manufatureiro proposto serão expostos ao decorrer das sessões seguintes.

Tabela 4-1. Dados referentes às máquinas.

	inProd	reset (u.t.)	tier
Máquina 0	2	10	0
Máquina 1	0	12	0
Máquina 2	0	8	1
Máquina 3	1	12	1
Máquina 4	2	17	1
Máquina 5	2	9	2
Máquina 6	0	4	2

Tabela 4-2. Relação de tempo de preparo (Setup) entre as peças na máquina 0.

Para / De	Peça 0	Peça 1	Peça 2
Peça 0	0	678	623
Peça 1	645	0	756
Peça 2	798	690	0

Tabela 4-3. Relação de tempo de preparo (Setup) entre as peças na máquina 1.

Para / De	Peça 0	Peça 1	Peça 2
Peça 0	0	598	756
Peça 1	678	0	636
Peça 2	899	672	0

Tabela 4-4. Relação de tempo de preparo (Setup) entre as peças na máquina 2.

Para / De	Peça 0	Peça 1	Peça 2
Peça 0	0	412	472
Peça 1	365	0	491
Peça 2	432	382	0

Tabela 4-5. Relação de tempo de preparo (Setup) entre as peças na máquina 3.

De \ Para	Peça 0	Peça 1	Peça 2
Peça 0	0	416	403
Peça 1	344	0	500
Peça 2	376	315	0

Tabela 4-6. Relação de tempo de preparo (Setup) entre as peças na máquina 4.

De \ Para	Peça 0	Peça 1	Peça 2
Peça 0	0	424	412
Peça 1	386	0	489
Peça 2	398	377	0

Tabela 4-7. Relação de tempo de preparo (Setup) entre as peças na máquina 5.

De \ Para	Peça 0	Peça 1	Peça 2
Peça 0	0	598	614
Peça 1	544	0	667
Peça 2	769	587	0

Tabela 4-8. Relação de tempo de preparo (Setup) entre as peças na máquina 6.

De \ Para	Peça 0	Peça 1	Peça 2
Peça 0	0	534	606
Peça 1	588	0	645
Peça 2	791	601	0

Tabela 4-9. Tempos de produção de cada peça em cada máquina de acordo com seu tier.

Máquina 0	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	12	8	10
Peça 1	10	6	11
Peça 2	15	3	12

Máquina 1	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	13	6	11
Peça 1	9	3	14
Peça 2	12	9	13

Máquina 2	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	10	8	11
Peça 1	12	4	10
Peça 2	18	6	12

Máquina 3	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	13	11	14
Peça 1	12	9	12
Peça 2	15	10	15

Máquina 4	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	15	8	10
Peça 1	14	9	12
Peça 2	15	6	12

Máquina 5	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	12	7	14
Peça 1	12	8	13
Peça 2	13	5	11

Máquina 6	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	17	12	9
Peça 1	19	16	12
Peça 2	21	12	15

Tabela 4-10. Dados referentes aos níveis de atuação das máquinas (tiers).

	size	tail
tier 0	2	0
tier 1	3	2
tier 2	2	5

4.2 Situação I

Dado o sistema introduzido na seção anterior, seja a abstração na figura 4-1 uma disposição do layout da fábrica. Como caso inicial, o sistema manufatureiro em teste terá sete máquinas,

três tiers e 14 lotes a serem produzidos. A tabela 4-11 fornece os dados dos lotes que estão aguardando produção.

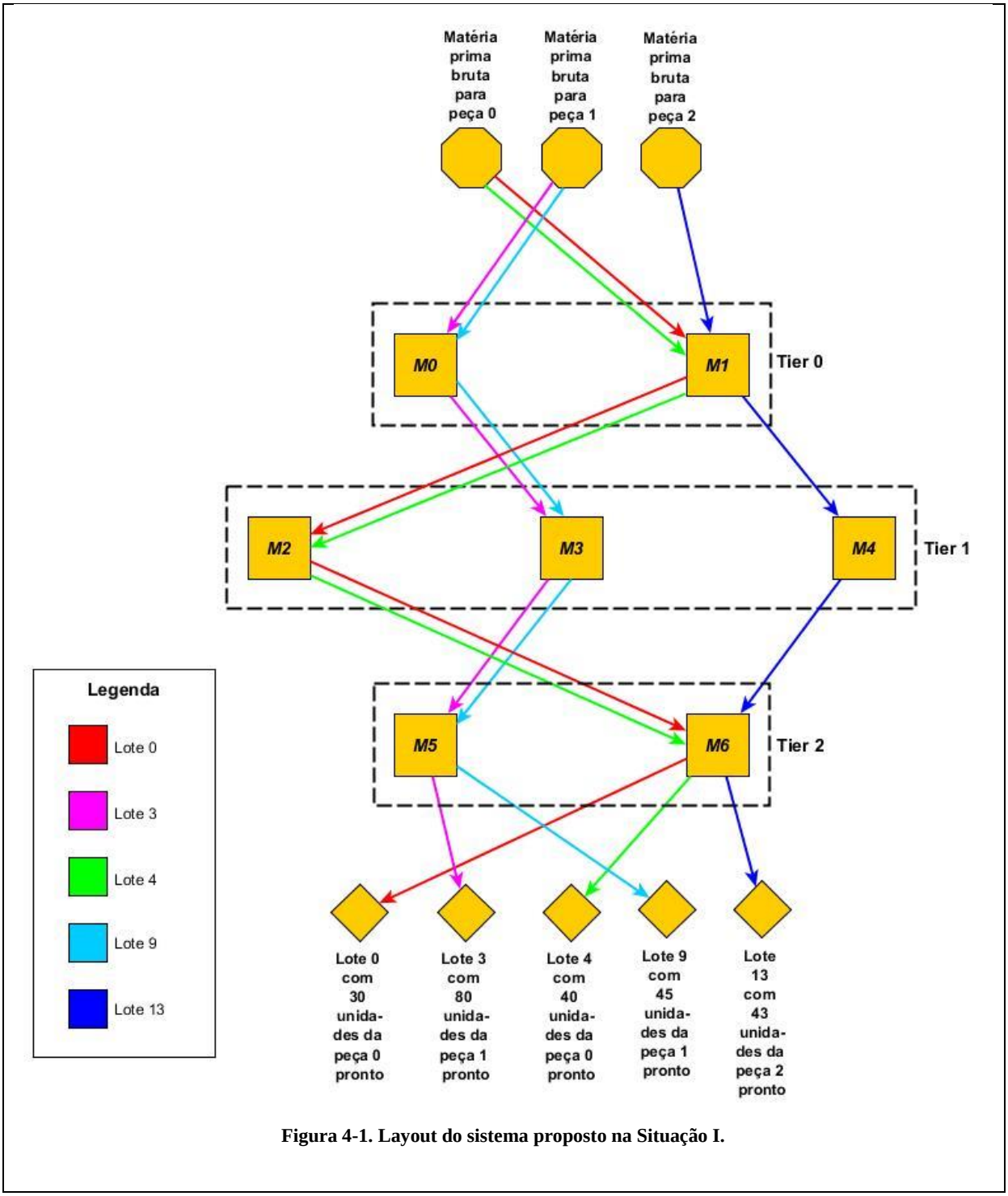

Figura 4-1. Layout do sistema proposto na Situação I.

Tabela 4-111. Dados referentes aos lotes a serem produzidos.

	prodType	prodAmount	deadline
Lote 0	0	30	2500 u.t.
Lote 1	1	50	7000 u.t.
Lote 2	2	45	4600 u.t.
Lote 3	1	80	7500 u.t.
Lote 4	0	40	3890 u.t.
Lote 5	0	30	3800 u.t.
Lote 6	2	35	6500 u.t.
Lote 7	0	20	7800 u.t.
Lote 8	0	45	15000 u.t.
Lote 9	1	45	12000 u.t.
Lote 10	2	60	9540 u.t.
Lote 11	1	22	4000 u.t.
Lote 12	0	51	10000 u.t.
Lote 13	2	43	9800 u.t.

Fazendo uso dos dados mostrados acima, faz-se a chamada ao escalonador CUDA o qual, seguindo o detalhamento feito no capítulo 3, busca o melhor escalonamento para a situação atual do sistema. A tabela 4-12 exibe o retorno emitido pelo escalonador o qual será passado ao módulo de interface e controle que, embora não implementado neste trabalho, trata as informações convertendo-as em instruções e as transmitindo ao módulo escalonador da linha de produção, para que então este configure as máquinas.

Tabela 4-12. Resultado retornado pelo escalonador CUDA na Situação I.

```
********** 1st TURN **********
Batch 0 (30 units of product 0)
        Starts in: 0 u.t.
        Finishes in: 1620 u.t.
        Looseness: 880 u.t.
        Scheduling path: Machine 1 --> Machine 2 --> Machine 6

Batch 3 (80 units of product 1)
        Starts in: 0 u.t.
        Finishes in: 6317 u.t.
        Looseness: 1183 u.t.
        Scheduling path: Machine 0 --> Machine 3 --> Machine 5
**********

********** 2nd TURN **********
Batch 4 (40 units of product 0)
        Starts in: 1620 u.t.
        Finishes in: 3780 u.t.
        Looseness: 110 u.t.
        Scheduling path: Machine 1 --> Machine 2 --> Machine 6
**********

********** 3rd TURN **********
Batch 13 (43 units of product 2)
        Starts in: 5400 u.t.
        Finishes in: 9600 u.t.
        Looseness: 200 u.t.
        Scheduling path: Machine 1 --> Machine 4 --> Machine 6
**********

********** 4th TURN **********
Batch 9 (45 units of product 1)
        Starts in: 6317 u.t.
        Finishes in: 9152 u.t.
        Looseness: 2848 u.t.
        Scheduling path: Machine 0 --> Machine 3 --> Machine 5
**********
```

```
****************** NON-PRODUCED BATCHES **************
Batch 1 (50 units of product 1)
***INFEASIBLE SCHEDULING!***
Batch 2 (45 units of product 2)
***INFEASIBLE SCHEDULING!***
Batch 5 (30 units of product 0)
***INFEASIBLE SCHEDULING!***
Batch 6 (35 units of product 2)
***INFEASIBLE SCHEDULING!***
Batch 7 (20 units of product 0)
***INFEASIBLE SCHEDULING!***
Batch 8 (45 units of product 0)
***INFEASIBLE SCHEDULING!***
Batch 10 (60 units of product 2)
***INFEASIBLE SCHEDULING!***
Batch 11 (22 units of product 1)
***INFEASIBLE SCHEDULING!***
Batch 12 (51 units of product 0)
***INFEASIBLE SCHEDULING!***
************************************************************
```

Os resultados mostrados na tabela 4-12 informam como deve ser feito o escalonamento do sistema na situação atual para que este atenda o maior número de lotes, dentro dos seus respectivos deadlines, no menor tempo possível.

Para entender o que se vê na tabela 4-12, tem-se que os turnos representam a ordem em que os lotes devem ser escalonados para produção, os dados logo abaixo do nome do lote representam, respectivamente: o instante em que o lote deve ser colocado em produção, o instante final da produção do lote, a folga com que este terminará sua produção mediante seu deadline e as máquinas que devem ser usadas para a produção do mesmo. A figura 4-1, além de abstrair o layout da fábrica, fornece também um diagrama de cores representando os caminhos seguidos pelos lotes que foram produzidos.

Os lotes que se encontram na sessão “NON-PRODUCED BATCHES” da tabela 4-12 são aqueles cujos escalonamentos são infactíveis na atual situação, i.e. não será possível produzi-los dentro de seus respectivos deadlines.

4.3 Situação II

Dado que, após o ocorrido durante a situação I, a fábrica fictícia receba novamente os mesmos pedidos de produção com exatamente as mesmas especificações. No entanto, as máquinas 2 e 3 apresentaram problemas e deverão ser desabilitadas, encontrando-se indisponíveis para o próximo escalonamento. A nova disposição entre máquinas e tiers está representada na figura 4-2.

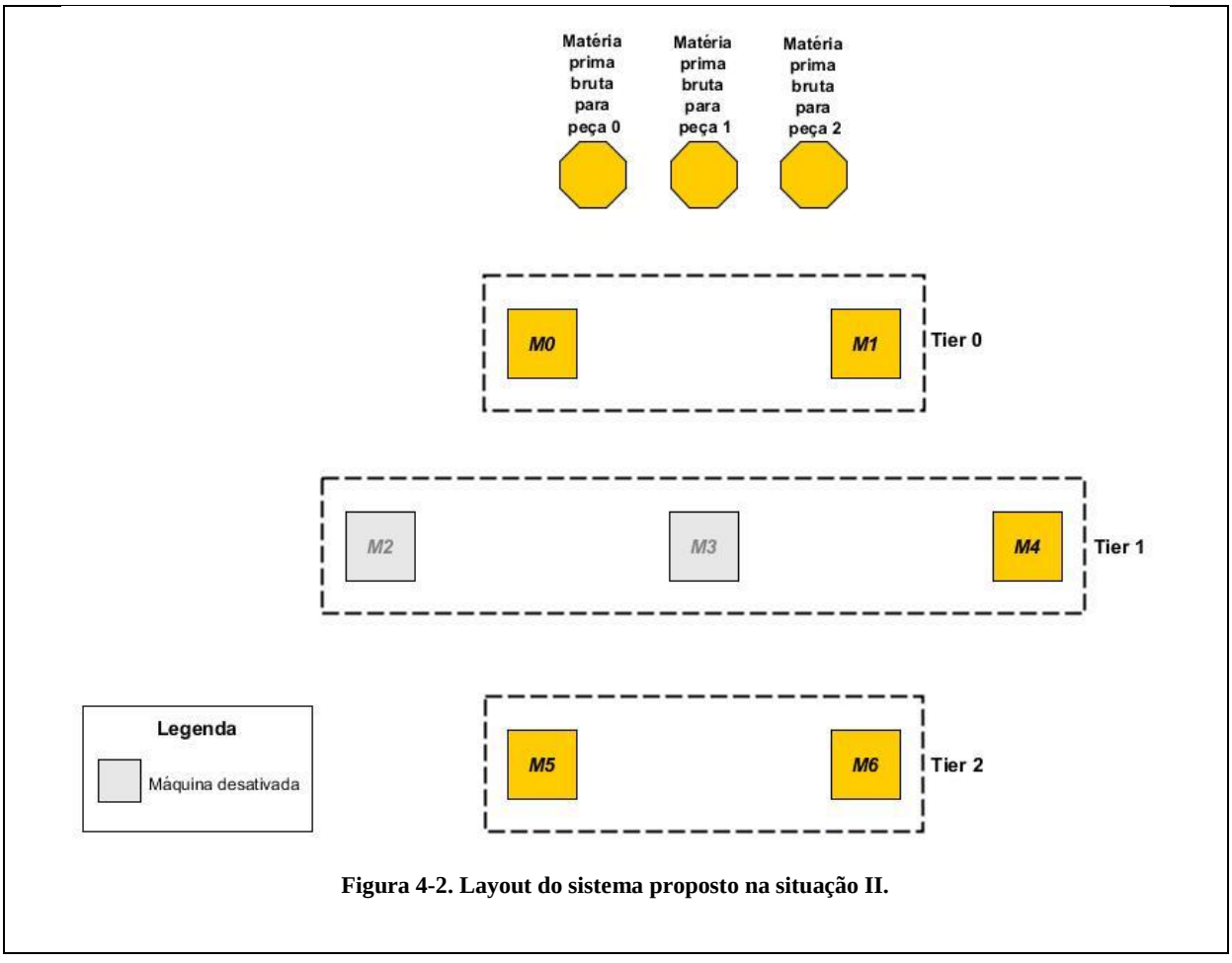

Figura 4-2. Layout do sistema proposto na situação II.

Dada a situação atual e sabendo dos resultados obtidos na situação I onde vários lotes não conseguiram ser atendidos, fez-se uso do módulo de usuário para gerar uma simulação de escalonamento com os dados atuais da linha de produção e dos lotes, de modo a prevenir um baixo desempenho em um estado de produção real. A tabela 4-13 mostra o resultado obtido pela simulação da situação II.

Tabela 4-13. Resultado retornado pelo escalonador CUDA na Situação II

```
********** 1st TURN **********
Batch 0 (30 units of product 0)
        Starts in: 0 u.t.
        Finishes in: 2288 u.t.
        Looseness: 212 u.t.
        Scheduling path: Machine 1 --> Machine 4 --> Machine 6
************************************************************

********** 2nd TURN **********
Batch 1 (50 units of product 1)
        Starts in: 2288 u.t.
        Finishes in: 6994 u.t.
        Looseness: 6 u.t.
        Scheduling path: Machine 1 --> Machine 4 --> Machine 6
************************************************************

********** 3rd TURN **********
Batch 8 (45 units of product 0)
        Starts in: 9282 u.t.
        Finishes in: 13754 u.t.
        Looseness: 1246 u.t.
        Scheduling path: Machine 0 --> Machine 4 --> Machine 6
************************************************************

********** NON-PRODUCED BATCHES **********
Batch 2 (45 units of product 2)
***INFEASIBLE SCHEDULING!***
Batch 3 (80 units of product 1)
***INFEASIBLE SCHEDULING!***
Batch 4 (40 units of product 0)
***INFEASIBLE SCHEDULING!***
Batch 5 (30 units of product 0)
***INFEASIBLE SCHEDULING!***
Batch 6 (35 units of product 2)
***INFEASIBLE SCHEDULING!***
Batch 7 (20 units of product 0)
***INFEASIBLE SCHEDULING!***
Batch 9 (45 units of product 1)
***INFEASIBLE SCHEDULING!***
```

```
Batch 10 (60 units of product 2)
***INFEASIBLE SCHEDULING!***
Batch 11 (22 units of product 1)
***INFEASIBLE SCHEDULING!***
Batch 12 (51 units of product 0)
***INFEASIBLE SCHEDULING!***
Batch 13 (43 units of product 2)
***INFEASIBLE SCHEDULING!***
**************************************************
```

Como o layout da fábrica foi alterado, a realização de uma prévia simulação demonstrou o que já era esperado: a ineficiência do sistema em atender às ordens de produção na atual situação.

4.4 Situação III

Dado a inviabilidade do sistema em atender a atual demanda, como fora concluído ainda na situação II, tem-se agora a inserção de cinco novas máquinas no layout da fábrica assim como mostra a figura 4-3. A escolha da disposição das novas máquinas na linha de produção se deu a partir de suas características como tempo de reset, de setup e de produção das peças, assim como mostra os dados contidos nas tabelas 4-14, 4-15 e 4-16 respectivamente.

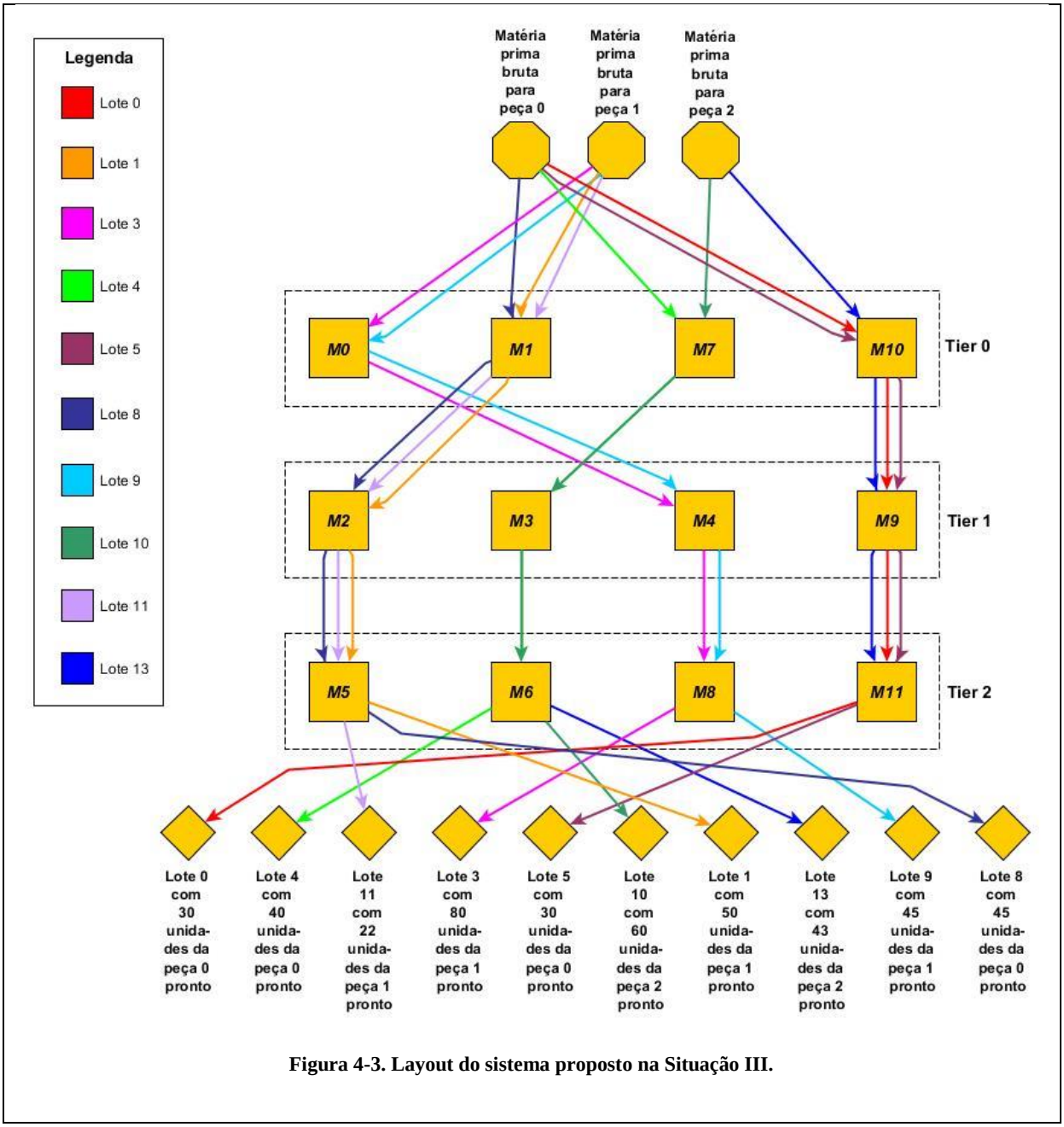

Figura 4-3. Layout do sistema proposto na Situação III.

Tabela 4-14. Novos dados referentes às máquinas.

	inProd	reset (u.t.)	tier
Máquina 0	2	10	0
Máquina 1	0	12	0
Máquina 2	0	8	1
Máquina 3	1	12	1
Máquina 4	2	17	1

Máquina 5	2	9	2
Máquina 6	0	4	2
Máquina 7	0	9	0
Máquina 8	0	13	2
Máquina 9	0	9	1
Máquina 10	0	4	0
Máquina 11	0	9	2

Assume-se que as novas máquinas sejam configuradas inicialmente, por padrão, para produzir a peça 0, daí explicando o motivo do campo "inProd" da tabela 4-14 conter o valor zero para as novas máquinas 7, 8, 9, 10 e 11.

Tabela 4-15. Relação de tempo de preparo (Setup) entre as peças nas novas máquinas.

Máquina 7	**Peça 0**	**Peça 1**	**Peça 2**
Peça 0	0	498	546
Peça 1	530	0	598
Peça 2	766	533	0

Máquina 8	**Peça 0**	**Peça 1**	**Peça 2**
Peça 0	0	500	546
Peça 1	534	0	578
Peça 2	743	512	0

Máquina 9	**Peça 0**	**Peça 1**	**Peça 2**
Peça 0	0	515	535
Peça 1	567	0	589
Peça 2	712	523	0

Máquina 10	**Peça 0**	**Peça 1**	**Peça 2**
Peça 0	0	508	566
Peça 1	511	0	562
Peça 2	731	523	0

Máquina 11	**Peça 0**	**Peça 1**	**Peça 2**
Peça 0	0	498	566
Peça 1	515	0	536
Peça 2	714	531	0

Tabela 4-16. Tempos de produção de cada peça em cada uma das novas máquinas de acordo com seu tier.

Máquina 7	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	15	9	16
Peça 1	13	11	14
Peça 2	18	10	13

Máquina 8	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	18	14	10
Peça 1	19	12	9
Peça 2	22	10	13

Máquina 9	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	14	10	12
Peça 1	15	9	8
Peça 2	19	9	13

Máquina 10	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	13	12	15
Peça 1	16	14	13
Peça 2	17	16	17

Máquina 11	**prodTime (tier 0)**	**prodTime (tier 1)**	**prodTime (tier 2)**
Peça 0	19	14	8
Peça 1	21	13	10
Peça 2	25	11	9

Tabela 4-17. Dados atualizados referentes aos níveis de atuação das máquinas (tiers).

	size	tail
tier 0	4	0
tier 1	4	4
tier 2	4	8

De acordo com os dados atualizados do sistema mostrados nas tabelas 4-14, 4-15, 4-16 e 4-17, o escalonador CUDA tem agora uma nova estruturação da linha de produção da fábrica e, portanto, precisa processar todos os novos dados para então gerar um novo escalonamento. O resultado deste novo escalonamento segue na tabela 4-18.

Tabela 4-18. Resultado retornado pelo escalonador CUDA na Situação III

```
********** 1st TURN **********
Batch 0 (30 units of product 0)
        Starts in: 0 u.t.
        Finishes in: 1590 u.t.
        Looseness: 910 u.t.
```

```
        Scheduling path: Machine 10 --> Machine 9 --> Machine 11

Batch 4 (40 units of product 0)
        Starts in: 0 u.t.
        Finishes in: 2744 u.t.
        Looseness: 1146 u.t.
        Scheduling path: Machine 7 --> Machine 3 --> Machine 6

Batch 11 (22 units of product 1)
        Starts in: 0 u.t.
        Finishes in: 2807 u.t.
        Looseness: 1193 u.t.
        Scheduling path: Machine 1 --> Machine 2 --> Machine 5

Batch 3 (80 units of product 1)
        Starts in: 0 u.t.
        Finishes in: 7007 u.t.
        Looseness: 493 u.t.
        Scheduling path: Machine 0 --> Machine 4 --> Machine 8
************************************************************

********** 2nd TURN **********
Batch 5 (30 units of product 0)
        Starts in: 1590 u.t.
        Finishes in: 3180 u.t.
        Looseness: 620 u.t.
        Scheduling path: Machine 10 --> Machine 9 --> Machine 11
************************************************************

********** 3rd TURN **********
Batch 10 (60 units of product 2)
        Starts in: 2744 u.t.
        Finishes in: 8379 u.t.
        Looseness: 1161 u.t.
        Scheduling path: Machine 7 --> Machine 3 --> Machine 6
************************************************************

********** 4th TURN **********
Batch 1 (50 units of product 1)
        Starts in: 2807 u.t.
```

```
        Finishes in: 5557 u.t.
        Looseness: 1443 u.t.
        Scheduling path: Machine 1 --> Machine 2 --> Machine 5
************************************************************

********************** 5th TURN **************************
Batch 13 (43 units of product 2)
        Starts in: 4770 u.t.
        Finishes in: 8888 u.t.
        Looseness: 912 u.t.
        Scheduling path: Machine 10 --> Machine 9 --> Machine 11
************************************************************

********************** 6th TURN **************************
Batch 9 (45 units of product 1)
        Starts in: 7007 u.t.
        Finishes in: 10067 u.t.
        Looseness: 1933 u.t.
        Scheduling path: Machine 0 --> Machine 4 --> Machine 8
************************************************************

********************** 7th TURN **************************
Batch 8 (45 units of product 0)
        Starts in: 8364 u.t.
        Finishes in: 12831 u.t.
        Looseness: 2169 u.t.
        Scheduling path: Machine 1 --> Machine 2 --> Machine 5
************************************************************

***************** NON-PRODUCED BATCHES ***************
Batch 2 (45 units of product 2)
***INFEASIBLE SCHEDULING!***
Batch 6 (35 units of product 2)
***INFEASIBLE SCHEDULING!***
Batch 7 (20 units of product 0)
***INFEASIBLE SCHEDULING!***
Batch 12 (51 units of product 0)
***INFEASIBLE SCHEDULING!***
************************************************************
```

Apesar de não ter sido possível atender à todos os lotes dentro de seus respectivos deadlines, a adição de novas máquinas na linha de montagem alcançou o comportamento esperado, i.e. foi possível escalonar e produzir uma quantidade consideravelmente maior do que na situação I, onde mais da metade os lotes não conseguiram ser produzidos dentro de seus respectivos deadlines. Ainda na figura 4-3, é possível observar o caminho percorrido por cada um dos lotes produzidos dentro do prazo estipulado.

Em todas as situações descritas anteriormente, os lotes que se encontram na seção "NON-PRODUCED BATCHES" (nas tabelas 4-12, 4-13 e 4-18) não necessariamente deixaram de ser produzidos, mas não puderam ser atendidos dentro de seus respectivos prazos (deadlines).

4.5 Desempenho

O desempenho é o principal motivo pela escolha de um modelo computacional executado sobre GPUs CUDA. Durante todo o decorrer desde trabalho, tentou-se mostrar o funcionamento e aplicação de tal modelo e, nesta seção, se dará a fundamentação da ferramenta escolhida.

Pelo fato de ser inviável criar, apresentar e abstrair um modelo com volumes consideráveis de dados da maneira como foi feita no decorrer deste trabalho, os tempos medidos durante as execuções do escalonador CUDA se encontram na casa dos milissegundos (10^{-3} s). No entanto, a escala não é tão interessante e sim o percentual que representa a diferença de desempenho entre os resultados.

Para concretizar a medição de desempenho, foi gerado um código na linguagem C semelhante ao do escalonador CUDA, porém fazendo as devidas alterações para que este execute na Central Processing Unit (CPU) encontrada no host. A execução deste código similar se dá de forma *sequencial*, i.e. usando uma única thread para executar os laços de repetições de busca de caminhos com escalonamento factível sem fazer qualquer uso de paralelismo. Os resultados desta comparação são representados nas tabelas 4-19, 4-20 e 4-21 para cada uma das situações apresentadas nas seções anteriores, respectivamente.

Tabela 4-19. Tabela de comparativo de desempenho entre os códigos do escalonador para GPU e para CPU na situação I.

	Tempo do código GPU (ms)	Tempo do código CPU (ms)		Desempenho da GPU em relação a CPU
Execução 1	0,866	0,739		- 17,18%
Execução 2	0,885	0,724		- 22,24%
Execução 3	0,853	1,117		+ 30,95%
Execução 4	0,859	0,727		- 18,16%
Execução 5	0,900	0,728		- 23,63%
Execução 6	0,865	1,149		+ 32,83%
Execução 7	0,697	0,723		+ 3,73%
Execução 8	0,871	0,726		- 19,97%
Execução 9	0,840	0,731		- 14,91%
Execução 10	0,863	0,740		- 16,62%
Média	$\frac{8,499}{10} = 0,8499$	$\frac{8,1045}{10} = 0,8104$		- 4,87%

Tabela 4-20. Tabela de comparativo de desempenho entre os códigos do escalonador para GPU e para CPU na situação II.

	Tempo do código GPU (ms)	Tempo do código CPU (ms)		Desempenho da GPU em relação a CPU
Execução 1	0,600	0,496		- 20,97%
Execução 2	0,491	0,490		- 0,2%
Execução 3	0,624	0,581		- 7,4%
Execução 4	0,614	0,497		- 23,54%
Execução 5	0,619	0,493		- 25,56 %
Execução 6	0,597	0,816		+ 36,68%
Execução 7	0,637	0,529		- 20,41%
Execução 8	0,545	0,501		- 8,78%
Execução 9	0,598	0,788		+ 31,77%
Execução 10	0,598	0,495		- 20,81%
Média	$\frac{5,923}{10} = 0,5923$	$\frac{5,686}{10} = 0,5686$		- 4,17%

Tabela 4-21. Tabela de comparativo de desempenho entre os códigos do escalonador para GPU e para CPU na situação III.

	Tempo do código GPU (ms)	Tempo do código CPU (ms)		Desempenho da GPU em relação a CPU
Execução 1	2,381	7,818		+ 228,35%
Execução 2	2,437	7,733		+ 217,32%
Execução 3	2,381	7,744		+ 225,24%
Execução 4	2,429	7,891		+ 224,86%
Execução 5	2,212	7,701		+ 248,14%
Execução 6	2,500	7,659		+ 206,36%
Execução 7	2,582	7,949		+ 207,86%
Execução 8	2,428	7,608		+ 213,34%
Execução 9	2,430	7,798		+ 220,9%
Execução 10	2,495	7,699		+ 208,58%
Média	$\frac{24,275}{10} = 2,4275$	$\frac{77.6}{10} = 7.76$		+ 219,74%

Como é possível observar, há uma grande diferença entre o desempenho obtido pela GPU nas tabelas 4-19 e 4-20 para aquele exibido na tabela 4-21. Tal diferença de desempenho pode ser explicada devido ao aumento da complexidade nos dados do problema na situação III e, principalmente, pelo problema de transferência de memória entre host e device no código elaborado para executar na GPU.

Todo código CUDA C, ao invocar a execução da função kernel (relembrando: aquela que executa na GPU), deve alocar memória na GPU e realizar a transferência de dados, inerentes ao processamento, do host para o device (GPU). Esta transferência de memória é um tanto custosa computacionalmente falando, o que gera uma queda de desempenho considerável quando um grande volume de dados deve ser transferido para a GPU e pouco processamento deve ser feita sobre estes dados – como é o caso das situações I e II.

Já na situação III, temos o acréscimo de cinco máquinas o que acarreta um aumento exponencial no processamento dos cálculos necessários para desenvolver o escalonamento, ao mesmo tempo em que o volume de dados transferidos entre host e device permanece praticamente o mesmo. Esta nova relação entre volume de processamento e transferência de dados favorece o processamento paralelo da GPU, o qual demonstra um desempenho muito

maior do que aquele apresentado pelo processamento sequencial da CPU. A falta de paralelismo na CPU exige que esta execute muito mais laços de repetição aninhados para chegar ao mesmo resultado que a GPU.

A verdadeira vantagem do uso das GPUs para se encontra na complexidade de processamento e não no volume de dados. Ou seja, a utilização das GPUs se torna vantajosa quando há uma complexidade exponencial de processamento sobre um volume de dados relativamente pequeno, como é o caso do planejamento de produção. Neste, as características do sistema (máquinas, funcionários, recursos, tipos de peças produzidas, lotes etc.) mudam muito pouco ou quase nada, no entanto o relacionamento entre todos estes dados crescem exponencialmente bem como a necessidade de processamento para tratar tal relacionamento.

Capítulo 5

Conclusões

Chegado ao final do presente trabalho, todas as informações levantadas ao longo deste, bem como o desenvolvimento do escalonador CUDA, têm por finalidade demonstrar a viabilidade e vantagens do uso de GPUs para processamento paralelo massivo tanto nos problemas de escalonamento encontrados na indústria manufatureira como em muitos outros setores que apresentem necessidades semelhantes.

Dado todo o desenvolvimento do escalonador aqui proposto, das situações apresentadas no caso de teste e dos dados coletados para medidas de desempenho, conclui-se que o paralelismo oferecido pelas GPUs CUDA é capaz de suportar altos níveis de complexidade de processamento, viabilizando a otimização de muitas aplicações que atualmente usam o paralelismo tradicional ou sequer usam qualquer paralelismo.

Conclui-se também que a solução proposta para os problemas de escalonamentos em sistemas de manufatura alcançou satisfatoriamente o objetivo buscado, isto é, a obtenção em tempo real de um novo escalonamento em resposta às alterações nas configurações da linha de produção de uma fábrica provando que, apesar de muito pouco utilizada atualmente, as GPUs são uma solução com excelente relação custo/benefício para o problema proposto.

5.1 Direções futuras

Buscando expandir a ideia aqui iniciada, pretende-se explicitar algumas direções para futuros trabalhos fazendo uso da tecnologia CUDA, a qual muito tem a oferecer e se encontra em constante atualização tornando-se cada vez mais conhecida e utilizada.

5.1.1 Controle de estoque e sistemas de manuseio de materiais

Neste trabalho foi estudado o uso da tecnologia das GPUs CUDA voltado para sistemas flexíveis de manufatura (FMS), entretanto seria interessante estender esta aplicação para os sistemas de controle de estoque e também para o tratamento dos problemas de elaboração dos sistemas manuseio de materiais (MHS) no chão da fábrica. Estes problemas fazem amplo uso de soluções computacionais para gerar modelos de estudo e aplicação.

5.1.2 Planejamento do layout da fábrica

Em conjunto com os problemas citados na sessão anterior e também com o simulador desenvolvido neste trabalho, a ideia de aplicar o ferramental oferecido pelas GPUs CUDA no planejamento do layout de uma fábrica, i.e. a organização e interligação das máquinas no chão da fábrica visando otimizar o aproveitamento do espaço físico e o fluxo de materiais na linha de produção, resultaria em um set de ferramentas poderoso podendo ser aplicado para resolver, se não todos, a grande maioria dos problemas de planejamento encontrados em uma indústria manufatureira.

Referências

1. NVIDIA CORPORATION. **Site de aplicações CUDA**, 2014. Disponivel em: <http://www.nvidia.com/object/gpu-applications.html>. Acesso em: 5 Junho 2014.

2. MANACERO JUNIOR, A. **Uma abordagem híbrida busca heurística/sistema especialista para sequenciamento de produção em sistemas de manufatura**. UNICAMP. Campinas, p. 14-21. 1991.

3. NEGAHBAN, A.; SMITH, J. S. Simulation for manufacturing system design and operation : literature review and analysis. **Journal of Manufacturing Systems**, v. 33, p. 241-261, 2014. ISSN ISSN: 0278-6125.

4. PITOMBEIRA NETO, A. R.; GONÇALVES FILHO, E. V. A simulation-based evolutionary multiobjective approach to manufacturing cell formation. **Computers & Industrial Engineering**, v. 59, p. 64-74, 2010. ISSN ISSN: 0360-8352.

5. KRISHNAN, M.; CHINNUSAMY, T. R.; KARTHIKEYAN, T. Performance study of flexible manufacturing system scheduling using dispatching rules in dynamic environment. **Procedia Engineering**, v. 38, p. 2793-2798, 2012. ISSN ISSN: 1877-7058.

6. BANASZAK, Z. A.; TANG, X. Q.; WANG, S. C. . Z. M. B. Logistics modes in flexible manufacturing. **Computers in Industry**, v. 43, p. 237-248, 2000. ISSN ISSN: 0166-3615.

7. PINEDO, M. L. **Scheduling:** theory, algorithms, and systems. 3. ed. ed. New York: Springer, 2008.

8. BLAZEWICZ, J.; DROR, M.; WEGLARZ, J. Mathematical programming formulations for machine scheduling: a survey. **European Journal of Operational Research**, v. 51, p. 283-300, 1991. ISSN ISSN: 0377-2217.

9. SANDERS, J.; KANDROT, E. **CUDA by example:** an introduction to general-purpose GPU programming. Ann Arbor, Michigan: Pearson Education, Inc., 2011; Fourth printing 2012.