CARLOS HENRIQUE CORRÊA TOLENTINO

MODELAGEM E ANÁLISE DE RESTRIÇÕES DE TEMPO REAL NO ESCALONAMENTO EM SÍNTESE DE ALTO NÍVEL

FLORIANÓPOLIS – SC
2004

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE PÓS-GRADUAÇÃO
EM CIÊNCIA DA COMPUTAÇÃO

CARLOS HENRIQUE CORRÊA TOLENTINO

MODELAGEM E ANÁLISE DE RESTRIÇÕES DE TEMPO REAL NO ESCALONAMENTO EM SÍNTESE DE ALTO NÍVEL

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Orientador:
LUIZ CLÁUDIO VILLAR DOS SANTOS

Florianópolis, Fevereiro de 2004

MODELAGEM E ANÁLISE DE RESTRIÇÕES DE TEMPO REAL NO ESCALONAMENTO EM SÍNTESE DE ALTO NÍVEL

Carlos Henrique Corrêa Tolentino

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação (Sistemas de Computação) e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____
Raul Sidnei Wazlawick (Coordenador)

Banca Examinadora

_____
Luiz Cláudio Villar dos Santos (Orientador)

_____
Luís Fernando Friedrich

_____
Olinto José Varela Furtado

_____
Rômulo Silva de Oliveira

Agradeço a Deus pela saúde e pelas oportunidades.

Agradeço à minha família, pelo apoio incondicional, pelo conforto em momentos difíceis, por se mostrar sempre otimista, confiante e, apesar da distância, presente.

Agradeço aos amigos de Palmas-TO, aos amigos do Quilombo, aos amigos do Morro das Pedras, aos amigos do LAPS, aos amigos do mestrado e tantos outros por terem me acompanhado e me dado força nessa jornada.

E um agradecimento especial ao Professor e amigo Luiz Cláudio, pela oportunidade e principalmente pela dedicação.

MODELAGEM E ANÁLISE DE RESTRIÇÕES DE TEMPO REAL NO ESCALONAMENTO EM SÍNTESE DE ALTO NÍVEL

Carlos Henrique Corrêa Tolentino

Fevereiro / 2004

Orientador: Luiz Cláudio Villar dos Santos.

Área de Conhecimento: Sistemas de Computação.

Palavras-chave: Síntese de Alto Nível, Escalonamento, Restrições de Tempo.

Número de Páginas: 70.

RESUMO:

Este trabalho apresenta a resolução de um problema clássico da Síntese de Alto Nível: o escalonamento sob restrições de recursos e de tempo. Para tanto utiliza uma abordagem orientada à exploração automática de soluções alternativas.

O problema consiste em escalonar as operações de um algoritmo buscando uma melhor utilização dos recursos físicos e satisfazendo uma série de restrições de recursos, de precedência e de tempo.

Os resultados experimentais mostram o sucesso das técnicas propostas em eliminar soluções de baixa qualidade do espaço de busca e melhorar a qualidade média do espaço de soluções. Em adição, na maioria dos testes realizados houve uma redução do tempo de busca por soluções de boa qualidade.

MODELLING AND ANALISYS OF REAL TIMING CONSTRAINTS ON HIGH LEVEL SINTHESYS SCHEDULING

Carlos Henrique Corrêa Tolentino

February / 2004

Advisor: Luiz Cláudio Villar dos Santos.

Area of Concentration: Computing Systems.

Keywords: High level synthesis, Scheduling, timing constraints.

Number of Pages: 70.

ABSTRACT:

This work presents a solution for a classical problem on High Level Sinthesys: timing and resources constrained scheduling. To this task it uses an aproach oriented to alternative solutions exploitation.

The problem consists of scheduling operations of an algorithm searching for a good utilization of phisical resources respecting the resources, precedence and timing constraints.

Experimental results shows that the porposed techniques successfully prunes the search space and increases the average quality of the solutions space. In adition, in the most of experiments, our technique provided lower search time for good solutions.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Gráficos

Lista de Algoritmos

1 Introdução

O mercado de produtos eletro-eletrônicos cresce cada vez mais. A maioria desses produtos utiliza circuitos integrados como componentes (telefones celulares, aparelhos de televisão, etc). Além disso, muitos desses produtos são sistemas que integram hardware e software na forma de *sistemas embutidos* ("embedded systems"). Um sistema embutido é qualquer dispositivo que contenha um computador programável e que não seja utilizado ou entendido como um computador de propósitos gerais [Wolf00]. Essencialmente, um sistema embutido é composto de processador, memória e, possivelmente, circuitos integrados de aplicação específica ou ASICs ("Application-Specific Integrated Circuits") [DeMicheli94].

Na produção de sistemas embutidos, a redução do tempo de projeto é um fator importante para aumentar a competitividade do produto. Outros fatores importantes são o consumo de energia e o tamanho do circuito. Para diminuir o tempo de projeto e o seu custo, o uso de ferramentas de CAD ("Computer-Aided Design") é mandatório. Tais ferramentas automatizam o projeto e são conhecidas também como ferramentas de EDA ("Electronic Design Automation").

O projeto de um sistema digital pode ser dividido em três fases: *modelagem*, *síntese* e *validação* [DeMicheli94]. A primeira consiste na transformação de uma idéia em um *modelo* que contém as funções que o circuito irá realizar. A síntese consiste no refinamento de um modelo mais abstrato, obtendo-se um modelo mais detalhado. A validação consiste na verificação da consistência entre o modelo mais abstrato e o modelo mais detalhado a partir dele obtido.

Tradicionalmente, os modelos são classificados em termos de *níveis de abstração* e segundo diferentes *visões* [Gajski87]. Consideramos três principais níveis de abstração. No *nível algorítmico* ou *arquitetural*, o sistema é caracterizado por um conjunto de operações, como processamento, transferência e armazenamento de dados. No *nível lógico*, o sistema é descrito através de um conjunto de funções lógicas. O *nível*

geométrico representa o sistema como um conjunto de entidades geométricas que formam o "layout" do circuito. A Figura 1.1 ilustra esquematicamente os níveis de abstração.

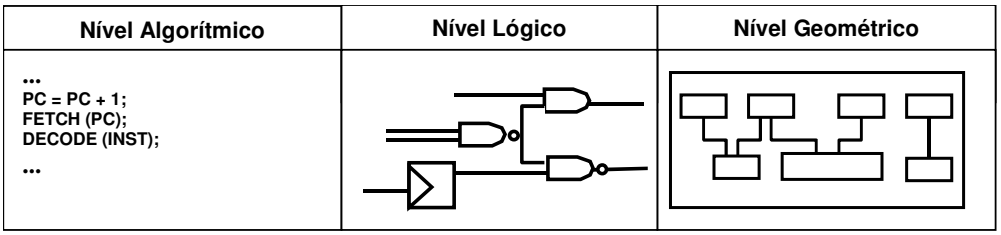

Figura 1.1: Níveis de abstração para representação de um sistema digital [DeMicheli94].

As diferentes visões são classificadas como *comportamental*, *estrutural*, e *física* [Gajski87]. A *visão comportamental* descreve a função do sistema sem considerar sua implementação, a *visão estrutural* descreve-o na forma de componentes e suas interconexões, e a *visão física* descreve seus objetos físicos, por exemplo, transistores e capacitores. A Figura 1.2 mostra uma descrição sob a visão comportamental e uma descrição correspondente sob a visão estrutural, ambas no nível arquitetural.

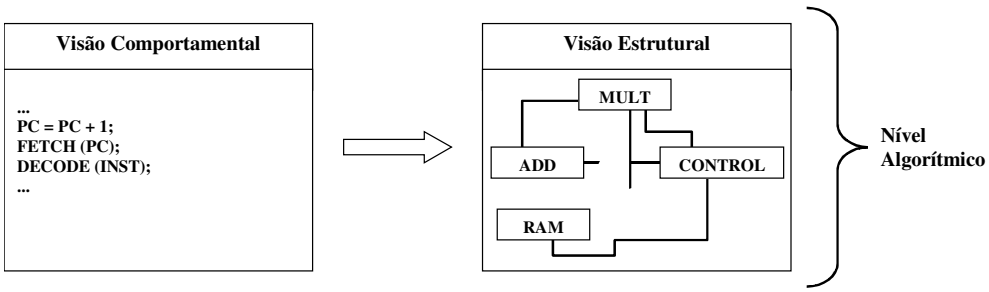

Figura 1.2: Visões comportamental e estrutural no nível arquitetural

A síntese do circuito pode ser vista como um conjunto de transformações entre duas visões axiais num mesmo nível de abstração, conforme ilustra a Figura 1.3. No nível algorítmico situa-se a *Síntese de Alto Nível*, que consiste na tradução de um modelo na visão comportamental, que apresenta as funções e operações do sistema (na forma de um algoritmo, por exemplo), para um modelo na visão estrutural, que apresenta operadores e suas interconexões. No nível lógico, a *Síntese Lógica* resulta na visão estrutural do circuito na forma de portas lógicas e suas interconexões, obtida a partir de uma descrição comportamental em termos de equações Booleanas, por exemplo.

Finalmente no nível físico, a *Síntese Física* utiliza a especificação dos componentes para definir o "layout" do circuito.

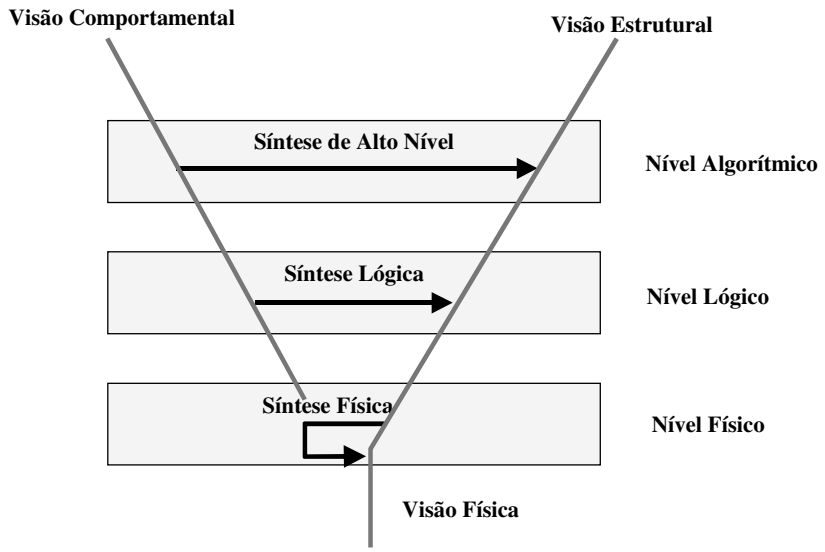

Figura 1.3: Níveis de abstração, visões e síntese [Gajski87].

Esta dissertação situa-se no contexto da Síntese de Alto Nível, doravante denominada simplesmente de Síntese, visto que é o único tipo de síntese aqui abordada. A Síntese resulta na arquitetura de um sistema digital, consistindo de uma *unidade operativa* ("datapath*")* e uma *unidade de controle* (*"*control unit*"*). O "datapath" é descrito como a interconexão de componentes tais como somadores, registradores, multiplicadores, etc. A unidade de controle é, geralmente, uma descrição simbólica de uma máquina de estados finitos.

A Síntese consiste de quatro etapas principais:

- *Seleção de módulos*, que determina que tipos de recursos são necessários no "datapath";
- *Alocação,* que determina a quantidade de recursos de cada tipo;
- *Escalonamento,* que determina quando as operações são executadas;
- *Ligação*, que associa cada operação a um recurso específico.

Os tipos de recursos (somadores, multiplicadores, etc.) são determinados pelo tipo de operações que serão realizadas (adição, multiplicação, etc.).

O trabalho de pesquisa descrito nesta dissertação foi realizado no âmbito do projeto "DESERT – Desenvolvimento de Ferramentas para Sistemas Embutidos sob Restrições de Tempo Real", dentro do grupo de trabalho "OASIS – Modelagem, Otimização e Síntese de **A**rquiteturas de **Sis**temas Digitais". No paradigma de modelagem adotado no projeto DESERT/OASIS, a descrição comportamental do sistema no nível algorítmico é traduzida para um grafo de fluxo de dados ou DFG ("Data Flow Graph"), no qual os vértices representam as operações. A unidade operativa é descrita na forma de um grafo do "datapath" ou DPG ("Data Path Graph"), que contém as especificações sobre tipos, quantidades e conexão entre os recursos do "datapath". A unidade de controle é modelada através de um grafo da máquina de estados ou SMG ("State Machine Graph"), onde cada vértice representa um estado da máquina ao qual estão associadas operações. A Figura 1.4 mostra esquematicamente a relação entre DFG, DPG e SMG.

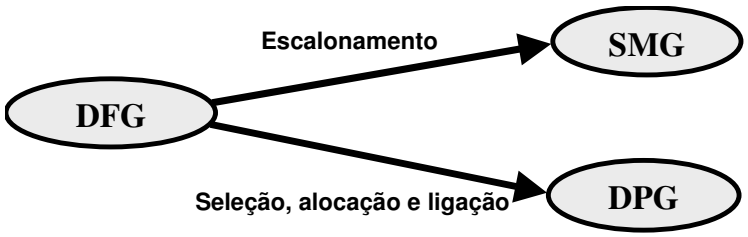

Figura 1.4: O processo de Síntese de Alto Nível sob o paradigma do projeto DESERT/OASIS.

À Síntese são impostas *restrições de projeto*, tais como o tempo máximo ou mínimo de execução (*restrição de tempo*), a ordem das operações a serem executadas (*restrição de precedência)* e o número de recursos disponíveis para executar as operações (*restrição de recursos*). A Síntese é guiada por uma série de *objetivos*, como a minimização da área do ASIC, consumo de energia, número de estados, etc.

Embora este trabalho concentre-se, sobretudo, nas *restrições de tempo*, também aborda *restrições de precedência* e de *recursos*. Uma restrição de precedência diz respeito à sucessão de operações, onde uma operação B consome um dado produzido por uma operação A, caracterizando uma *dependência de dados*. Dessa forma a operação A deve preceder a operação B. Restrições de recursos são relacionadas à quantidade de cada tipo de operadores disponível no "datapath". Restrições de tempo podem ser vistas de duas formas: entre as operações, quando existe uma relação temporal entre duas

operações, o que chamamos de *atraso relativo*, e restrição de tempo global ou de *latência*, que diz respeito ao tempo total de execução de todas as operações.

Restrições de precedência são modeladas no DFG através de arestas, indicando que a operação representada pelo vértice de origem da aresta deve preceder a operação representada pelo vértice destino. Restrições de recursos são modeladas no DPG a partir do resultado da seleção de módulos, da alocação e da ligação das operações presentes no DFG. As restrições de precedência são extraídas da descrição algorítmica através de um processo de compilação, conforme mostra a Figura 1.5.

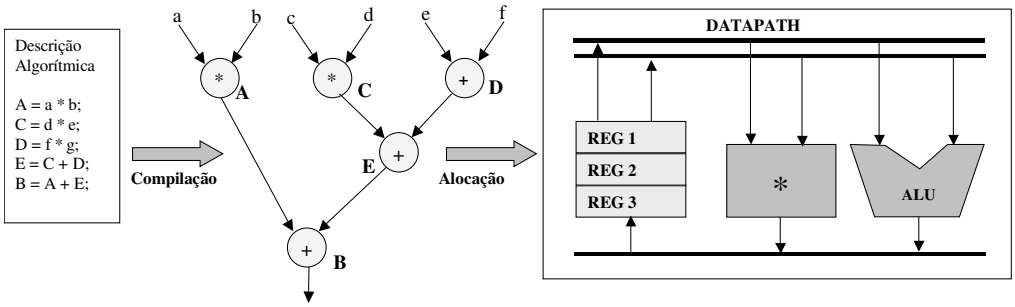

Figura 1.5 Captura de restrições de precedência e de recursos.

No exemplo acima observamos, tanto na descrição algorítmica quanto no DFG, dois tipos de operações: adição e multiplicação. Cada tipo de operação é relacionado a um tipo diferente de recurso. Adições são associadas a um recurso do tipo *Unidade Lógico-Aritmética* (ALU – "Arithmetic Logical Unit") e multiplicações são associadas a um recurso do tipo multiplicador. Note que a restrição de recursos imposta é de um recurso de cada tipo.

Esta dissertação aborda a modelagem e a análise de restrições de tempo para a resolução de um problema clássico de síntese: o escalonamento sob restrições de tempo e de recursos. Embora a modelagem seja formalizada no Capítulo 2, ela é a seguir introduzida informalmente através de exemplos.

A Figura 1.6a ilustra um DFG onde foram anotadas, além das restrições de precedência, restrições de atraso relativo e de latência, ambas representadas pelos valores associados às arestas, o que chamamos de *pesos* das arestas. A Figura 1.6b ilustra um SMG obtido,

via escalonamento, a partir do DFG da Figura 1.6a, assumindo como restrição de recursos que haja um operador de cada tipo.

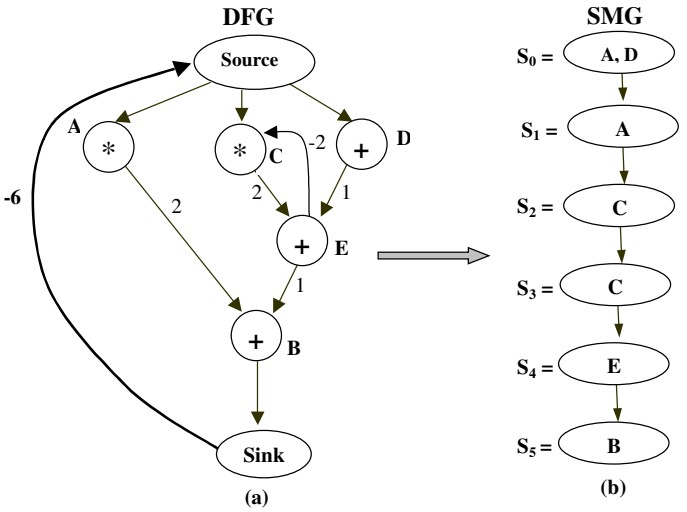

Figura 1.6 Um exemplo da influência de restrições de tempo no escalonamento.

Para modelar o fato de que o multiplicador leva dois ciclos de relógio para executar uma multiplicação, às arestas (C, E) e (A, B) foi atribuído o peso 2. Da mesma forma, para indicar que a ALU leva apenas um ciclo para executar uma soma, às arestas (D, E) e (E, B) foi atribuído o peso 1. À quantidade de ciclos que uma operação leva para ser executada chamamos de *atraso de execução*.

Nesta dissertação, adota-se a modelagem de restrições de tempo proposta em [DeMicheli94], assim definida:

- Uma aresta com origem em uma operação v_1, destino em uma operação v_2 e peso +d indica que a operação v_2 deve ser executada, no mínimo, d ciclos depois de v_1, o que chamamos de *restrição de atraso mínimo*. Um exemplo é a aresta (E, B) da Figura 1.6a.
- Uma aresta com origem em uma operação v_2, destino em uma operação v_1 e com peso –d, indica que a operação v_2 deve ser executada, no máximo, d ciclos depois de v_1, o que chamamos de *restrição de atraso máximo*. Um exemplo desse tipo de restrição é a aresta (E, C) da Figura 1.6a.
- Dadas duas operações v_1 e v_2 o par de arestas (v_1, v_2) e (v_2, v_1) com pesos de, respectivamente, d e –d, significa que os atrasos máximo e mínimo entre as operações são iguais, ou seja, |d|, o que caracteriza uma *restrição de atraso exato*

entre as operações. Como exemplo de atraso exato, observe as operações C e E na Figura 1.6a e o par de arestas (C, E) e (E, C).

Na Figura 1.6a, os vértices Source (fonte) e Sink (sumidouro) modelam, respectivamente, as entradas e saídas primárias do sistema. A aresta em destaque, com origem em Sink e destino em Source, representa uma restrição de latência. No exemplo, o valor –6 significa que o atraso máximo entre as operações Source e Sink é de seis ciclos, ou seja, o atraso total de todas as operações situadas entre Source e Sink deve ser de, no máximo, seis ciclos.

Uma vez que cada estado do SMG tem tempo de duração de um ciclo, observando a Figura 1.6 percebe-se que o escalonamento do DFG da Figura 1.6a deve resultar num SMG com no máximo seis estados (s_0, ..., s_5)**.** Neste caso, dizemos que a solução para o problema de escalonamento sob restrições recursos e de tempo apresentada no SMG da Figura 1.6b satisfaz a restrição de latência, ou seja, é *factível*.

Tornando a restrição de latência mais severa, percebemos que a combinação entre restrição de recursos e de tempo pode levar a uma situação em que não existe escalonamento possível que satisfaça a todas as restrições. Por exemplo, ajustando a restrição de latência para cinco ciclos, tornamos a solução apresentada na Figura 1.6b *infactível*, fazendo-se necessário verificar se há uma solução alternativa factível. A Figura 1.7b mostra uma solução alternativa que satisfaz a restrição de latência de cinco ciclos, mantendo os mesmos recursos. A diferença entre as soluções está na ordem em que as operações foram escalonadas.

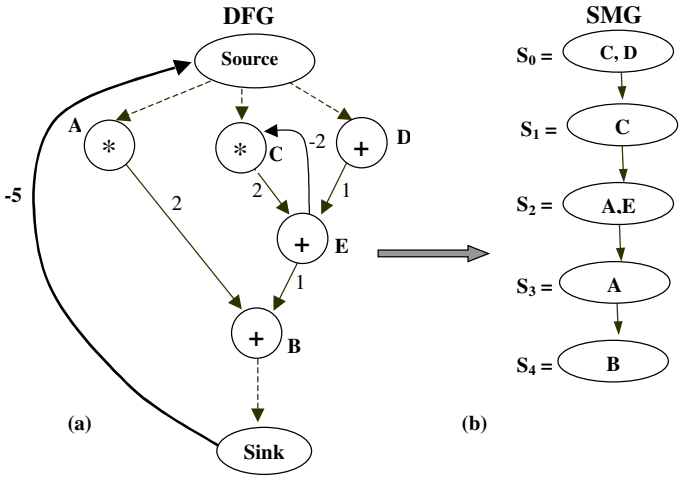

Figura 1.7: Uma solução alternativa para restrição mais severa.

Os dois principais objetivos do trabalho de pesquisa aqui descrito são:

- **Detecção de infactibilidade**: as restrições de tempo são analisadas de forma a detectar o mais cedo possível, antes ou durante o escalonamento, se uma solução é infactível.
- **Melhoria do espaço de busca**: As restrições de tempo são utilizadas para modificar dinamicamente o escalonamento, de forma a eliminar soluções inferiores, melhorando a qualidade média das soluções no espaço de busca e, conseqüentemente, aumentando a probabilidade de se encontrar a solução ótima em menor tempo.

Esta dissertação está organizada da seguinte forma:

O Capítulo 2 traz a modelagem e formulação dos problemas. Esse capítulo serve de fundamentação matemática para amparar as técnicas propostas.

O Capítulo 3 inicia com uma discussão de trabalhos correlatos que, além de situar a proposta desta dissertação no contexto da área de pesquisa, busca justificar a contribuição deste trabalho em face do estado da arte. Em seguida descreve como o Analisador de Restrições, objeto deste trabalho, insere-se na abordagem de síntese adotada no Projeto DESERT/OASIS.

O Capítulo 4 apresenta os resultados experimentais desta pesquisa, interpretando-os e discutindo-os.

O Capítulo 5 resume as principais contribuições deste trabalho de pesquisa a partir das quais são apresentadas conclusões e perspectivas de continuidade.

As referências bibliográficas encontram-se no Capítulo 6.

2 Modelagem e Formulação do Problema

Este capítulo descreve os modelos utilizados na representação do problema e de sua solução. Inicialmente, são apresentados os modelos que capturam comportamento e estrutura. Em seguida, descreve-se a modelagem de restrições. Ao final, utilizando-se a modelagem de comportamento, estrutura e restrições, os problemas abordados são formalizados.

2.1 Modelagem de Comportamento e Estrutura

Como mencionado anteriormente, o resultado da Síntese é uma arquitetura de um sistema digital consistindo de um "datapath" e de uma unidade de controle, obtidas a partir de uma descrição comportamental, representada por um DFG. O "datapath" é representado por um DPG e a unidade de controle por um SMG. Os tipos de grafos utilizados na modelagem são formalmente apresentados a seguir.

Definição 2.1: Um *grafo polar de fluxo de dados* DFG(V, E, W) é um grafo orientado ponderado onde cada vértice $v_i \in V$ representa uma operação, onde cada aresta $(v_i, v_j) \in E$ representa uma dependência de dados entre as operações v_i e v_j e onde o peso $w_{ij} \in W$ relativo à aresta (v_i, v_j) representa o atraso relativo entre as operações. Os pólos são os vértices v_0 e v_n, denominados fonte e sumidouro, respectivamente.

Definição 2.2: Um *grafo polar da máquina de estados* SMG(S, T) é um grafo orientado onde cada vértice $s_i \in S$ representa um estado e onde cada aresta $(s_i, s_j) \in T$ representa uma transição entre os estados s_i e s_j.

Definição 2.3: Um *grafo polar de unidade operativa* é um DPG(C, W) é um grafo orientado onde cada vértice $c_i \in C$ representa um componente e cada aresta $(c_i, c_j) \in W$ representa uma interconexão entre os componentes c_i e c_j.

O escalonamento sob restrições de recursos necessita de um parâmetro relativo à quantidade e os tipos de recursos disponíveis, de um mecanismo que associe as operações a um determinado tipo de operador e ainda de um mecanismo que associe operações a estados do SMG. Na seqüência são formalizadas as noções relativas ao escalonamento sob restrição de recursos.

Definição 2.4: Um *vetor de restrição de recursos* **a** é um vetor onde cada componente a_k representa o número de unidades funcionais de um determinado tipo $k \in \{1, 2, ..., n\}$.

Definição 2.5: Uma função τ: $V \rightarrow \{1, 2, ..., n\}$ é uma função que mapeia cada operação para um tipo de recurso $k \in \{1, 2, ..., n\}$.

Definição 2.6: O *atraso de execução* d_i é um valor correspondente ao número de ciclos necessários para completar a execução de uma operação v_i em um recurso do tipo $\tau(v_i)$.

Definição 2.7: Uma função denominada *escalonamento* φ: $V \rightarrow S$ é uma função que mapeia cada operação v_i do DFG para um estado $s_i = \varphi(v_i)$ do SMG, tal que:

- $\forall (v_i, v_j) \in E$: $(s_k = \varphi(v_i)\ s_r = \varphi(v_j)) \Rightarrow r \geq k + w_{ij}$ e
- $| \{v_i : \tau(v_i) = p \text{ e } k \leq m < k + d_i\} | \leq a_p$, para cada tipo de recurso $p = 1, 2,..., N$ e para cada estado s_m, com $m = 1, 2, ..., M$.

Note que o primeiro item da definição acima se refere às restrições de precedência e o segundo as restrições de recursos.

Definição 2.8: Dados um SMG(S,T) e um estado arbitrário $s_k \in S$, o conjunto das *operações prontas* no estado s_k, denotado por A_k, é o conjunto de todas as operações v_i, ainda não escalonadas tais que, para todo predecessor v_j de v_i, tal que $s_n = \varphi(v_j)$ e $n + w_{ji} \leq k$.

Nem todas as operações podem ser executadas num único ciclo. Uma operação v_i com $d_i > 1$ é denominada uma operação *multiciclo*. Quando uma operação multiciclo é escalonada num estado s_k, ela não pode ser completada neste estado, tornando-se uma operação *pendente* no estado subseqüente s_{k+1}.

Definição 2.9: Dados um SMG(S,T) e um estado arbitrário $s_k \in S$, o conjunto das *operações pendentes*, denotado por U_k, no estado s_k, é o conjunto de todas as operações v_i, já escalonadas em estados anteriores, tais que $s_n=\varphi(v_i)$ e $n+d_i > k$.

Se uma operação multiciclo é escalonada num estado s_k, na transição para o estado s_{k+1} a sua execução ainda não foi terminada. Surge, então, a noção de *atraso remanescente de execução*, formalizado como segue.

Definição 2.10: Dados um estado s_k e uma operação $v_i \in V$, tal que $s_p = \varphi(v_i)$ o *atraso remanescente de execução* $dr(v_i,s_k)$ é o número de ciclos ainda necessários para completar a execução de v_i em um recurso do tipo $\tau(v_i)$, ou seja, $(p + d_i) - k$.

As técnicas de modelagem e análise de restrições de tempo a serem descritas no Capítulo 3 baseiam-se nas noções de caminho e distância, conforme formalizado a seguir.

Definição 2.11: Um *caminho* em um grafo orientado G(V, E) com início no vértice v_i e término no vértice v_n, é uma seqüência finita de vértices posicionados de 0 até n, tal que $(v[i-1], v[i]) \in E$ e $i \in \{1, 2, ..., n\}$.

Definição 2.12: Dado um caminho $(v_0,...,v_i,...,v_j,...,v_n)$ em um grafo orientado G(V, E) com início no vértice v_0 e término no vértice v_n, um *sub-caminho* é um caminho com início no vértice v_i e término no vértice v_j tal que $0 \le i \le j \le n$.

Definição 2.13: Um *ciclo* em um grafo orientado G(V, E), é um caminho com início no vértice v_0 e término no vértice v_n, tal que $v_0 = v_n$.

Definição 2.14: Dado um grafo orientado G(V, E) e dois vértices arbitrários v_i e $v_j \in V$, diz-se que o vértice v_i *alcança* v_j através de p, escrito $v_i \xrightarrow{p} v_j$, se há um caminho p de v_i até v_j. Muitas vezes esse caminho não precisa ser nomeado, escrito $v_i \xrightarrow{*} v_j$. Note que este caminho pode ser trivial, no caso $v_i = v_j$.

Definição 2.15: Dados um DFG(V, E, W) e um caminho p tal que $v_i \xrightarrow{p} v_j$ a *distância* $d(v_i,v_j)$ percorrida de v_i a v_j no caminho p é igual à soma dos pesos das arestas pertencentes a p.

Definição 2.16: Dado um caminho $p = (v_0,v_1,...,v_n)$ em um grafo orientado G(V, E), seu *comprimento* l(p) é a distância percorrida entre os vértices v_0 e v_n.

As arestas de um caminho no SMG representam transições entre estados e não são ponderadas. Considerando que cada estado dura exatamente um ciclo de relógio, assume-se que as transições têm peso unitário. Assim, a *distância* percorrida em um SMG é simplesmente o número de arestas percorridas.

Definição 2.17: Dados um SMG(S, T) e um caminho p tal que $s_i \xrightarrow{p} s_j$ a *distância* $d(s_i,s_j)$ percorrida de s_i a s_j no caminho p é igual ao número de arestas de p.

Definição 2.18: Dados um DFG(V, E, W), um SMG(S, T) e um vértice $v_i \in V$ escalonado no estado $s_k \in S$, o *tempo de início* de v_i é $t_i = d(s_0, s_k)$.

Definição 2.19: Dado um grafo orientado G(V,E,W) e dois vértices arbitrários v_i e $v_j \in V$ a *distância máxima* entre v_i e v_j, denotada por $d_{max}(v_i ,v_j)$, é dada por:
$\forall\ p \mid v_i \xrightarrow{p} v_j : \text{Max} \{ d(v_i,v_j) \}$.

Definição 2.20: Dados um grafo orientado G(V,E,W) e dois vértices arbitrários v_i e $v_j \in V$ tais que $v_i \xrightarrow{lp} v_j$, o *caminho mais longo* (“longest path”) de v_i a vj, é o caminho em que a distância percorrida é máxima.

Um ponto interessante a ser ressaltado é que um sub-caminho (v_i,v_j) do caminho mais longo (v_0,v_n) é o caminho mais longo entre v_i e v_j, tal que $0 \le i \le j \le n$ [Cormen90].

Definição 2.21: A *latência* λ de um SMG(S, T) é o número de estados n pertencentes a ao caminho mais longo entre o vértice fonte s_o e o vértice sumidouro s_n.

2.2 Modelagem de Restrições de Tempo Real

O tema principal desta dissertação é a modelagem e análise de restrições de tempo na fase de escalonamento. Essas restrições são essencialmente atrasos relativos entre operações. Os tipos de atraso (mínimo, máximo e exato) são definidos a seguir.

Definição 2.22: Dados um DFG(V, E, W) e um número $d \in Z^+$, um *atraso de no mínimo* d ciclos entre duas operações v_i e $v_j \in V$, é modelado através de uma aresta $(v_i,v_j) \in E$ com peso $w_{ij} = +d$, indicando que a operação v_j deve iniciar sua execução no mínimo d ciclos após a operação v_i ter iniciado sua execução.

A Figura 2.1 ilustra um exemplo de atraso mínimo.

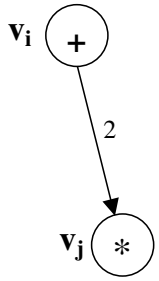

Figura 2.1: Atraso mínimo entre as operações v_i e v_j.

Definição 2.23: Dados um DFG(V, E, W) e um número $d \in Z^+$, um *atraso de no máximo* d ciclos entre as operações v_i e $v_j \in V$ é modelado através de uma aresta $(v_j, v_i) \in E$ com peso $w_{ji} = -d$, indicando que a operação v_j deve iniciar execução no máximo d ciclos após a operação v_i ter iniciado sua execução.

A Figura 2.2 ilustra um exemplo atraso máximo.

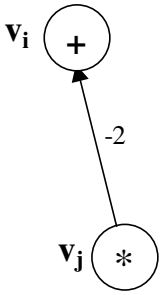

Figura 2.2: Atraso máximo entre as operações v_i e v_j.

Note que se o atraso mínimo coincide com o atraso máximo, então o atraso entre as operações é *exatamente* d ciclos, o que pode ser representado combinando as modelagens das Definições 2.22 e 2.23, como segue.

Definição 2.24: Dados um DFG(V, E, W) e um número $d \in Z^*$, um *atraso de exatamente* d ciclos entre duas operações v_i e $v_j \in V$ é modelado através de uma aresta $(v_i, v_j) \in E$ com peso $w_{ij} = +d$ e de uma aresta $(v_j, v_i) \in E$ com peso $w_{ji} = -d$, indicando que a operação v_j deve iniciar execução exatamente d ciclos após a operação v_i ter iniciado sua execução.

A Figura 2.3 ilustra um exemplo de atraso exato entre as operações v_i e v_j.

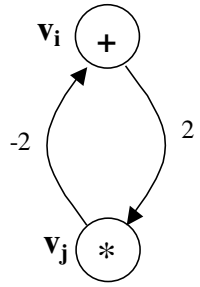

Figura 2.3: Atraso exato entre as operações v_i e v_j.

Definição 2.25: Dado um DFG(V, E ,W), uma *restrição de latência* tal que o número de ciclos gastos para executar todas as operações em V não exceda o valor máximo λ_{max} é modelada como um atraso de no máximo λ_{max} ciclos entre o vértice fonte v_0 e o vértice sumidouro v_n.

2.3 O Problema de Escalonamento

Como visto anteriormente a fase de escalonamento pode ser definida informalmente como a ordenação da execução de operações ao longo do tempo, respeitando as dependências de dados e a disponibilidade dos recursos. No contexto deste trabalho, segundo a Definição 2.7, cada operação de um DFG quando escalonada, é mapeada para um estado de um SMG.

2.3.1 Escalonamento sob Restrições de Recursos

A formulação abaixo pressupõe que, a partir de uma descrição comportamental obtém-se um DFG(V, E) onde os vértices representam as operações e as arestas representam as dependências de dados. Assume-se também que, no fluxo de projeto a seleção e a alocação de componentes do "datapath" precedem o escalonamento. Conseqüentemente, aquelas etapas impõem ao escalonamento uma restrição de recursos.

O escalonamento é um problema clássico de otimização [DeMicheli94], onde o objetivo é obter uma solução que minimize a latência do SMG, respeitando as restrições de precedência e de recursos. O problema é formalmente apresentado como segue:

Problema 2.1 - Dado um grafo de fluxo de dados DFG(V,E), e um vetor de restrição de recursos **a**, encontre um escalonamento φ que minimize a latência λ.

Como este problema é intratável no caso geral, a busca de uma solução exata levaria a tempos de execução proibitivos. Por isso, heurísticas costumam ser usadas em sua solução.

2.3.2 Escalonamento sob Restrições de Recursos e de Tempo

Em alguns domínios de aplicação, além da restrição de recursos, podem ser impostas restrições de atraso relativo e de latência. Nestes casos, a descrição comportamental também é traduzida para um grafo de fluxo de dados, porém na forma DFG(V,E,W) onde W é o conjunto dos pesos das arestas, que representam as restrições de tempo.

Neste caso o escalonamento deve garantir que todas as operações sejam executadas respeitando as restrições de precedência, de recursos e de tempo, conforme formalizado a seguir.

Problema 2.2 - Dado um grafo de fluxo de dados DFG(V,E,W), onde os pesos W representam restrições de tempo, e um vetor de restrição de recursos **a**, encontre um escalonamento que satisfaça simultaneamente as restrições de tempo, de precedência e de recursos.

Note que não se trata de um problema de otimização, pois não há função custo a ser otimizada. Este é um problema de factibilidade, pois o objetivo é determinar se existe ou não a possibilidade de escalonar todas as operações respeitando simultaneamente todas as restrições.

É importante ressaltar que os Problemas 2.1 e 2.2 são intratáveis [DeMicheli94] [Timmer93]. Quando o número de recursos é ilimitado, o Problema 2.2 degenera em um problema particularmente interessante, assim enunciado:

Problema 2.3: Dado um grafo de fluxo de dados DFG(V, E, W), onde os pesos W representam restrições de tempo, encontre um escalonamento que satisfaça as restrições de precedência e de tempo.

A ausência de restrições de recursos resulta em um problema tratável. Como mostrado em [DeMicheli94], algoritmos clássicos como o de Bellman-Ford [Cormen90] e de Liao-Wong [Liao-Wong83] resolvem o Problema 2.3 garantindo exatidão em tempo polinomial.

O tema central desta dissertação é o Problema 2.2. Embora as técnicas de análise de restrições de tempo sejam desenvolvidas para resolvê-lo, elas podem ser aplicadas ao Problema 2.1, transformando-o no Problema 2.2. Com restrições de tempo adicionadas ao Problema 2.1, este tem seu espaço de soluções reduzido, tornando a busca de soluções mais eficiente.

Note que, para um mesmo DFG(V, E, W), se é detectada infactibilidade para o Problema 2.3, também será detectada para o Problema 2.2. Esta propriedade será utilizada para verificar infactibilidade antes de se resolver o Problema 2.2.

No próximo capítulo, mostra-se como a modelagem de restrições de tempo aqui discutida pode ser utilizada não somente para capturar restrições de tempo real e verificar sua factibilidade, mas também para influenciar o processo de escalonamento, melhorando a qualidade das soluções geradas.

3 Descrição da Abordagem e de sua Extensão

Este capítulo primeiramente discute trabalhos correlatos e, em seguida, resume a abordagem proposta em [Santos98]. Por fim, o capítulo mostra a extensão daquela abordagem de forma a capturar e explorar restrições de tempo.

3.1 Trabalhos Correlatos

3.1.1 Abordagens Exatas

Em relação à resolução do problema de escalonamento, existem abordagens que utilizam métodos exatos e abordagens que utilizam métodos heurísticos. Um exemplo de método exato é a ILP ("Integer Linear Programming"), ou Programação Linear Inteira [DeMicheli94], na qual a modelagem do problema é feita utilizando-se variáveis de decisão binárias com dois índices, onde o primeiro indica a operação e o segundo, o instante de escalonamento. Restrições de recursos e de tempo são modeladas através de sistemas de inequações simultâneas. Esta técnica ampara-se na disponibilidade de pacotes de programas de propósitos gerais e oferece facilidade de incorporação de restrições adicionais.

Em [Timmer93][Timmer95] apresenta-se uma técnica de escalonamento sob restrições de recursos e de tempo. Essa técnica reformula o problema de escalonamento para recair num problema clássico, conhecido como "Bipartite Graph Matching". A reformulação baseia-se nas noções de *intervalo de execução das operações*, denotado como OEI ("Operation Execution Interval") e *intervalo de execução dos módulos*, denotado por MEI ("Module Execution Interval").

Esta técnica constrói um grafo bipartido contendo os vértices correspondentes a cada OEI de um lado e os vértices correspondentes a cada MEI do outro. O grafo resultante é denominado de BSG ("Bipartite Schedule Graph"). A técnica consiste em eliminar arestas do BSG que não pertencem a um "matching". Isto resulta numa redução dos OEIs, diminuindo o espaço de busca. A infactibilidade do problema é detectada quando

não existe “matching” para o BSG. Apesar dos bons resultados [Timmer95], a técnica baseia-se numa abordagem exata para um problema intratável. Portanto, não há como garantir sua eficiência para DFGs com grande número de vértices.

Em [DeMicheli94] sugere-se a modelagem de restrições de tempo através de arestas ponderadas de um DFG (tal como apresentado na Seção 2.2). Essa modelagem permite o uso do Algoritmo de Bellman-Ford [Cormen90] para obter uma solução para o problema de escalonamento sob restrições de tempo (Problema 2.3), ou para detectar sua infactibilidade. Em grafos onde o número de restrições de atraso máximo (Definição 2.23) é reduzido, sugere-se a utilização do Algoritmo de Liao-Wong [Liao-Wong83], devido à sua mais baixa complexidade computacional para grafos esparsos. A simples aplicação desses algoritmos para resolver o Problema 2.2, ignoraria as restrições de recursos.

A modelagem sugerida em [DeMicheli94] é estendida em [Mesman97] para amparar suas técnicas de analise de restrições. Todas as restrições consideradas (tempo, precedência, recursos, latência, etc) são modeladas num mesmo grafo, chamado de SFG (“Sequence Flow Graph”), através de arestas chamadas de arestas de seqüência. Quando se detecta que operações entram em conflito pela utilização de um recurso, uma aresta de seqüência é inserida entre essas operações, reduzindo sua liberdade de escalonamento, e conseqüentemente, o espaço de busca.

A eficiência dessa abordagem depende do quão severas são as restrições. Quanto mais severas, mais cedo uma solução factível é encontrada. Uma limitação dessa abordagem é que ela visa a obtenção de *uma* solução factível o mais cedo possível, porém não leva o projetista a explorar outros aspectos do projeto. Por exemplo, pode existir uma solução factível não explorada que requeira menor quantidade de registradores, ou que resulte em menor consumo de energia do que a solução encontrada anteriormente.

3.1.2 Abordagens Heurísticas

A literatura está repleta de abordagens heurísticas. Um dos métodos mais conhecidos é o “List Scheduling”, descrito em [DeMicheli94], que pode ser usado para minimização da latência do escalonamento dada uma restrição de recursos e vice-versa. Tem baixa

complexidade computacional, porém não garante uma solução ótima. Outro método conhecido é o "Force Directed Scheduling" [DeMicheli94], que apresenta resultados melhores do que o "List Scheduling", mas resulta num maior tempo de execução devido à sua maior complexidade.

Segundo Heijligers [Heijligers95], a desvantagem do "List Scheduling" é a utilização de heurísticas gulosas para estabelecer a prioridade entre as operações. Para superar essa desvantagem, algoritmos genéticos foram utilizados para gerar as prioridades, resultando numa convergência em direção à melhor solução. Entretanto, aquele trabalho não considera simultaneamente restrições de recursos e de tempo.

3.1.3 Discussão

A principal vantagem das técnicas exatas é a garantia de uma solução ótima. A desvantagem, como já visto, é o seu elevado custo de execução, resultado da alta complexidade. A maioria das abordagens em que se utilizam técnicas heurísticas levam a uma única solução nem sempre satisfatória.

Não restringir o espaço de busca a uma única solução pode representar uma boa alternativa na resolução do problema de escalonamento sob restrições severas. Uma abordagem orientada à exploração automática de soluções alternativas, proposta em [Santos98] é adotada em [Azambuja02] para escalonamento e alocação de registradores sob fluxo de execução seqüencial e condicional e em [Ferrari02] na paralelização de laços de iteração.

Essa abordagem, que é resumida a seguir, é também adotada nesta dissertação e estendida de forma que a modelagem e a análise de restrições de tempo sejam incorporadas à resolução do problema de escalonamento sob restrições de recursos (Problema 2.1) transformando-o no problema de escalonamento sob restrições de recursos e de tempo (Problema 2.2). Assim, procura-se manter um equilíbrio entre o tempo computacional e a qualidade das soluções encontradas.

3.2 Abordagem Orientada à Exploração Automática de Soluções Alternativas

A abordagem adotada baseia-se na exploração do espaço de busca de soluções utilizando *codificações de prioridades*. Prioridades são relevantes na construção de uma solução se, num dando instante, o número de operações prontas para escalonamento é maior do que a quantidade de recursos disponíveis. Assim deve-se ter um critério de escolha, o que leva à noção de prioridade. A prioridade relativa entre operações pode ser codificada através de uma permutação Π das operações do DFG.

Se duas operações A e B competem por um mesmo recurso e a operação A precede a operação B na permutação Π, A tem prioridade na utilização do recurso. Quando uma operação é escolhida para escalonamento em detrimento de outra dizemos que foi tomada uma *decisão de escalonamento*.

A abordagem pode ser decomposta em dois blocos principais chamados *Explorador* e *Construtor*, conforme ilustra a Figura 3.1. O Explorador é responsável por gerar as codificações de prioridade. Na interação entre os blocos, uma permutação Π, gerada pelo Explorador, é passada como parâmetro ao Construtor que, a partir desta, faz o escalonamento das operações gerando uma solução. O custo da solução é calculado e retornado ao Explorador. Diferentes permutações resultam em soluções possivelmente diferentes, com custos possivelmente distintos.

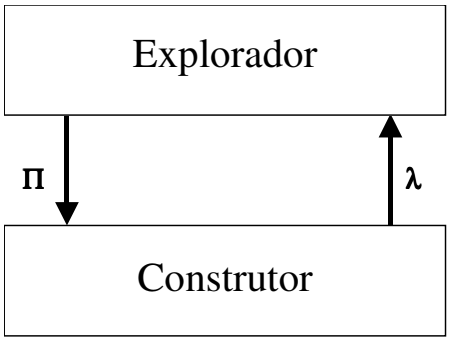

Figura 3.1: A decomposição da abordagem.

A diferença entre as soluções está no fato de que as operações podem ser escalonadas em ordem diferente dependendo da codificação de prioridade. Assim o bloco Explorador avalia o *custo* de cada solução gerada pelo Construtor com a finalidade de encontrar a melhor delas. Para o Problema 2.1, o custo é a latência λ do escalonamento.

Uma vez que os blocos são independentes, mantendo-se a interface de comunicação entre eles, um pode ser modificado sem interferir no funcionamento do outro, isto é, desde que se mantenha o uso de permutações como parâmetro de entrada do Construtor e o custo da solução como valor de retorno, o algoritmo de escalonamento pode ser modificado de forma a otimizar esse processo sem interferir na geração das codificações de prioridades e vice-versa.

O bloco Explorador não faz parte do escopo deste trabalho. Um tópico a ser abordado no grupo de trabalho OASIS está relacionado à incorporação de *meta-heurísticas* para melhorar o processo de exploração de soluções. Atualmente, o Explorador limita-se a gerar permutações aleatoriamente.

O bloco Construtor é decomposto em dois blocos, o *Paralelizador* e o *Escalonador*, conforme a Figura 3.2.

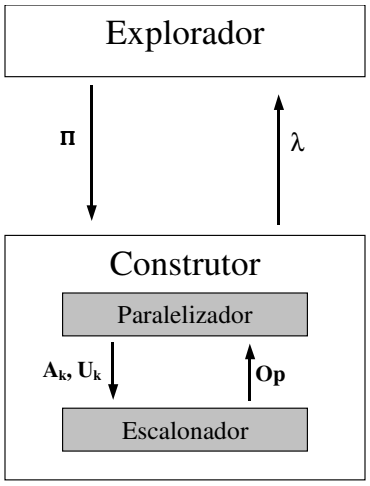

Figura 3.2: A decomposição do bloco Construtor.

3.2.1 O Paralelizador

O bloco Paralelizador é responsável por detectar a possibilidade de execução simultânea de operações, buscando deixar sempre o maior número de operações prontas para escalonamento. Esse bloco deve ser capaz de encontrar paralelismo inclusive através de construções condicionais [Azambuja02] e de laços de iteração [Ferrari02]. Por

simplicidade, mas sem perda de generalidade, este trabalho limita-se a tratar algoritmos com fluxo de execução puramente seqüencial.

Como dito anteriormente, o resultado do escalonamento é um SMG, onde a cada estado são associadas operações do DFG. O Paralelizador tem como tarefa criar cada estado s_k do SMG, um conjunto A_k de operações prontas para escalonamento nesse estado, e um conjunto U_k de operações pendentes nesse estado, passados como parâmetros ao Escalonador.

3.2.2 O Escalonador

O Escalonador observa a disponibilidade de recursos, a codificação de prioridade e os conjuntos A_k e U_k para escalonar o maior número possível de operações e retorna ao Paralelizador um conjunto OP_k de operações escalonadas no estado s_k. Esse processo se repete até que tenham sido escalonadas todas as operações.

O algoritmo proposto por Aiken, Nicolau e Novak [Aiken95] pode ser facilmente adaptado para explorar soluções alternativas [Santos98] [Azambuja01]. Uma vez que é fácil adaptá-lo para suportar restrições de tempo, esse algoritmo de escalonamento é também adotado nesta dissertação.

3.3 Extensão da Abordagem

A abordagem original [Santos98] [Azambuja02] [Ferrari02] está apta a resolver somente o problema de escalonamento sob restrição de recursos (Problema 2.1). Para resolver o problema de escalonamento sob restrições de recursos e de tempo (Problema 2.2) um outro bloco é inserido: o *Analisador de Restrições*.

Antes de iniciar a construção de soluções, o Analisador é responsável por verificar a infactibilidade do problema de escalonamento, através da análise das restrições de tempo capturadas pelos pesos das arestas do DFG.

Durante a construção de uma solução, o Analisador anota dinamicamente no DFG as decisões de escalonamento tomadas, através da inserção de novas arestas ponderadas, que impõem restrições de tempo adicionais às operações ainda não escalonadas, o que

pode resultar na infactibilidade da solução. Por isso, o Analisador é responsável ainda por verificar dinamicamente a infactibilidade da solução que está sendo construída.

A Figura 3.3 ilustra a integração do Analisador à abordagem original.

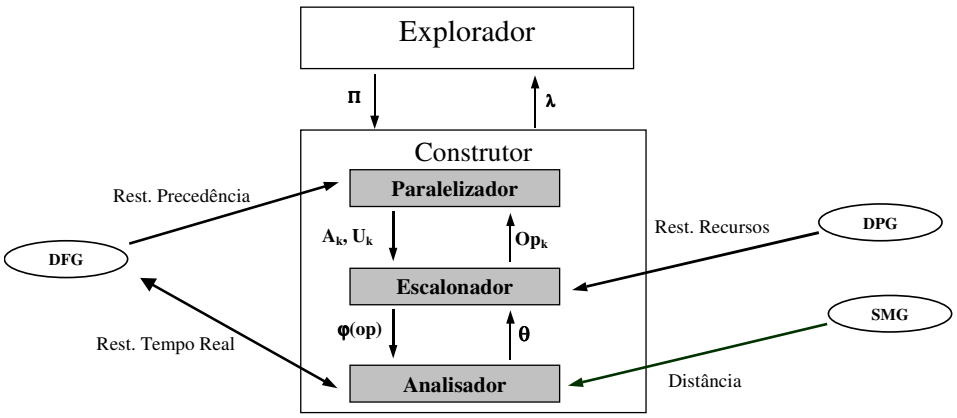

Figura 3.3: Extensão da abordagem.

A interação entre Analisador e Escalonador continua até que todas as operações sejam escalonadas e a solução resulte factível (θ = verdadeiro) ou até ser detectada sua infactibilidade (θ = falso), fazendo com que o Escalonador aborte a construção da solução para uma dada codificação de prioridade Π.

3.4 O Analisador de Restrições

Esta importante seção descreve detalhadamente o Analisador de Restrições, onde residem as principais contribuições desta dissertação.

Inicialmente, descreve-se a modelagem dinâmica de decisões de escalonamento e a captura do impacto das restrições de recursos. Em seguida, discute-se a verificação de infactibilidade em suas duas vertentes: estática e dinâmica, e são apresentadas duas técnicas para verificação de infactibilidade.

Ao final, mostra-se como a detecção da infactibilidade de uma solução pode ser usada para melhorar a qualidade das próximas soluções geradas, através da assim chamada técnica de recuperação de soluções.

3.4.1 Modelagem Dinâmica de Decisões de Escalonamento

Nesta seção são examinadas algumas conseqüências da modelagem de restrições de tempo adotada (veja Definições 2.22, 2.23, 2.24 e 2.25), ilustrando como elas podem ser utilizadas para modelar dinamicamente as decisões tomadas durante o escalonamento. A Figura 3.4 ilustra como são modeladas decisões de escalonamento.

De acordo com as Definições 2.22 e 2.23, a Figura 3.4a indica que a operação B deve ser executada no mínimo 2 ciclos de relógio e no máximo 4 ciclos de relógio após a operação A ter iniciado sua execução. Portanto, há *liberdade de escalonamento* para a operação B e cabe ao escalonador decidir em que ciclo de relógio ela será efetivamente escalonada. Por outro lado, conforme a Definição 2.24, a Figura 3.4b indica que a operação B deve ser executada exatamente dois ciclos de relógio após A ter iniciado sua execução, não havendo, portanto, qualquer liberdade de escalonamento.

Em particular, quando um dos vértices envolvidos é o vértice-fonte, tal modelagem pode ser utilizada para representar decisões tomadas pelo escalonador, pois o vértice-fonte serve como referência de tempo. Seja t_i o tempo em que o vértice v_i inicia sua execução (Definição 2.18). Para o vértice-fonte v_0, tem-se $t_0 = 0$.

Considere o exemplo da Figura 3.4c. Os pesos das arestas indicam que a operação C pode iniciar execução nos instantes $t_C = 3$, $t_C = 4$ ou $t_C = 5$. O fato de a operação C poder iniciar execução em diferentes instantes de tempo sugere a noção de *mobilidade*, conforme formalizado abaixo:

Definição 3.1: Seja um vértice arbitrário v_i de um DFG(V, E ,W) cujo vértice-fonte é v_0, tal que $(v_0, v_i) \in E$ e $(v_i, v_0) \in E$. A *mobilidade* μ_i do vértice v_i é a distância percorrida no ciclo (v_0, v_i, v_0).

Note na Figura 3.4c que a mobilidade da operação C é 2, significando que há 3 ($\mu_C + 1$) alternativas para escaloná-la. Suponhamos agora que se decida escalonar C no instante $t_C = 3$. Esta decisão pode ser anotada no DFG conforme ilustra a Figura 3.4d. Note que, por força do escalonamento, a mobilidade da operação C foi fixada em zero,

significando que uma única alternativa foi escolhida, não havendo, portanto, qualquer liberdade de escalonamento remanescente.

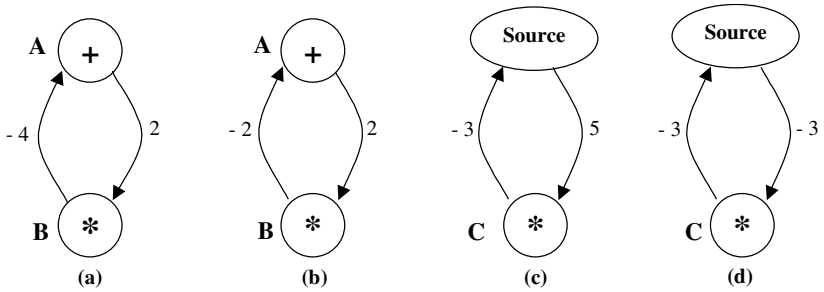

Figura 3.4: Modelagem de restrições de tempo real.

Assim, na medida em que as operações são escalonadas, restrições de tempo podem ser anotadas no DFG para influenciar futuras decisões do escalonador. Quando uma operação v_i é escalonada em um instante de tempo $t_i = d$, uma aresta (v_0, v_i) com peso d e uma aresta (v_i, v_0) com peso –d são inseridas, caracterizando uma restrição do tipo atraso exato. A Figura 3.5 ilustra restrições de tempo anotadas no DFG durante um escalonamento, além do SMG resultante.

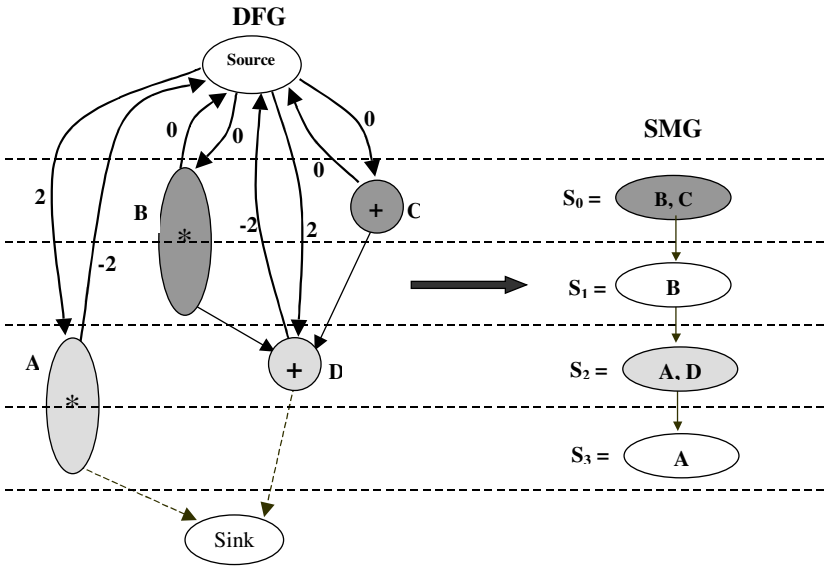

Figura 3.5: Exemplos de modelagem de decisões de escalonamento.

Como nos exemplos anteriores, supõe-se que as operações de multiplicação levam dois ciclos para serem executadas e as operações de adição apenas um ciclo. Para simplificar a Figura 3.5, os pesos das arestas que representam as dependências de dados foram

omitidos, já que o objetivo é enfatizar as restrições de tempo anotadas durante o escalonamento. Para melhor entendimento, cores diferentes foram associadas a diferentes decisões de escalonamento. Observe que o valor das restrições anotadas corresponde ao estado em que as operações foram escalonadas. Logo, para os sucessores do vértice Source escalonados no primeiro estado as arestas de atraso máximo e mínimo têm peso zero.

3.4.2 O Impacto das Restrições de Recursos

Uma vez que às operações também são impostas restrições de recursos, tais restrições têm influencia determinante sobre o custo das soluções. A seguir apresenta-se como tal influência é capturada.

Como já dito anteriormente, num estado s_k podem existir mais operações prontas para escalonamento do que unidades funcionais disponíveis para executá-las. O critério de escolha da operação que será executada é a prioridade. As operações escolhidas são fixadas nesse estado, enquanto as demais somente serão executadas em estados subseqüentes.

O fato de uma operação não poder ser escalonada em um estado devido à carência de recursos pode tornar a solução infactível. Como o mecanismo de verificação de infactibilidade é baseado na análise das restrições de tempo, é natural que restrições de tempo sejam utilizadas para refletir o impacto das restrições de recursos, a fim de que tais restrições possam ser interpretadas pelo Analisador.

Para indicar que uma operação será executada posteriormente, uma restrição de atraso mínimo é inserida. O peso associado a essa restrição pode definir a precisão com que a restrição de recursos será refletida, e conseqüentemente o quanto contribui para detectar infactibilidade.

Uma operação pronta não escalonada no estado atual s_k, será escalonada, no mínimo no estado s_{k+1}. Uma restrição de atraso mínimo, representado por uma aresta (v_0,v_i) com $w_{0i} = k+1$, seria suficiente para representar a falta de recursos.

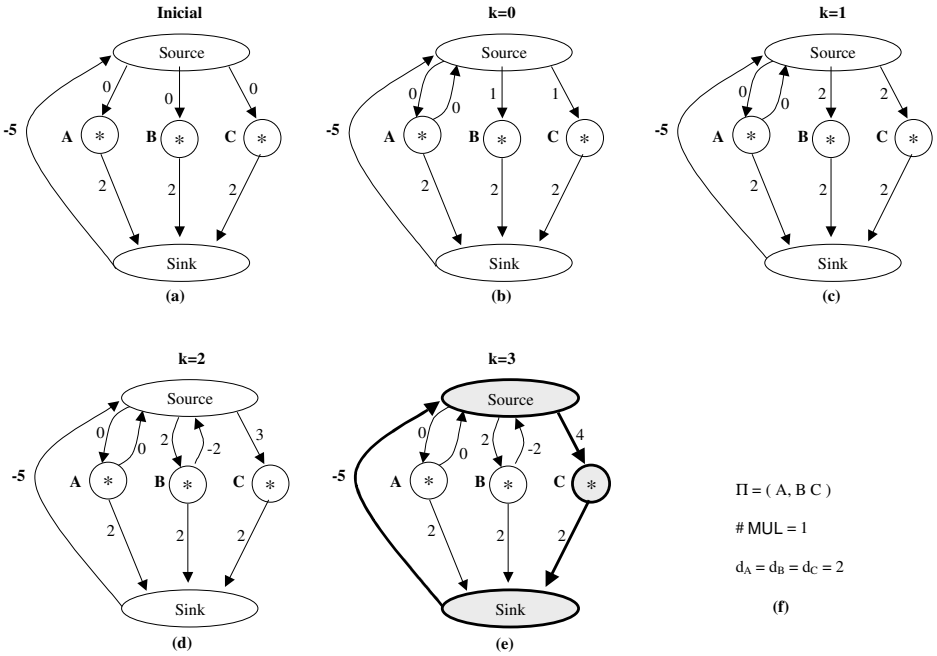

Figura 3.6: Restrições de tempo refletindo as restrições de recursos.

Considere o exemplo da Figura 3.6, que ilustra a transformação do DFG ao longo do escalonamento de quatro estados. Observe que, antes do escalonamento ser iniciado (Figura 3.6a) todas as operações estão prontas para serem escalonadas no estado s_0. Tendo A prioridade sobre as demais, ela é escalonada no estado s_0 enquanto às operações B e C são impostas restrições de tempo (Figura 3.6b). No estado s_1, B e C são novamente analisadas e, em vista à carência de recursos livres, suas restrições de tempo são novamente atualizadas (Figura 3.6c). Como a operação A é finalizada no estado s_1, a operação B é escalonada no estado s_2 (Figura 3.6d). O escalonamento continua até que no estado s_3 (Figura 3.6e), é detectada a infactibilidade da solução.

Pela simplicidade do exemplo, pode-se perceber que três operações com atraso de dois ciclos necessitariam de seis ciclos para serem executadas num único multiplicador. Considerando o exemplo da Figura 3.6, não há solução factível. Quanto antes for detectado esse fato, menos tempo será gasto pesquisando soluções não satisfatórias.

Assumindo a existência de operações multiciclo, a modelagem proposta no exemplo anterior já não reflete com precisão o impacto das restrições de recursos. Uma modelagem mais precisa, baseada no atraso remanescente de execução das operações (Definição 2.10) é apresentada a seguir.

Considere o exemplo da Figura 3.7, onde o DFG é o mesmo do exemplo anterior. Com todas as operações prontas (Figura 3.7a), a operação A, por ter maior prioridade, é escalonada (Figura 3.7b). Uma vez que $dr(A,s_0) = 2$, às operações B e C são impostas restrições de tempo com peso igual a $k+ dr(A,s_0)$, indicando que elas deverão esperar exatamente dois ciclos até que um recurso esteja livre.

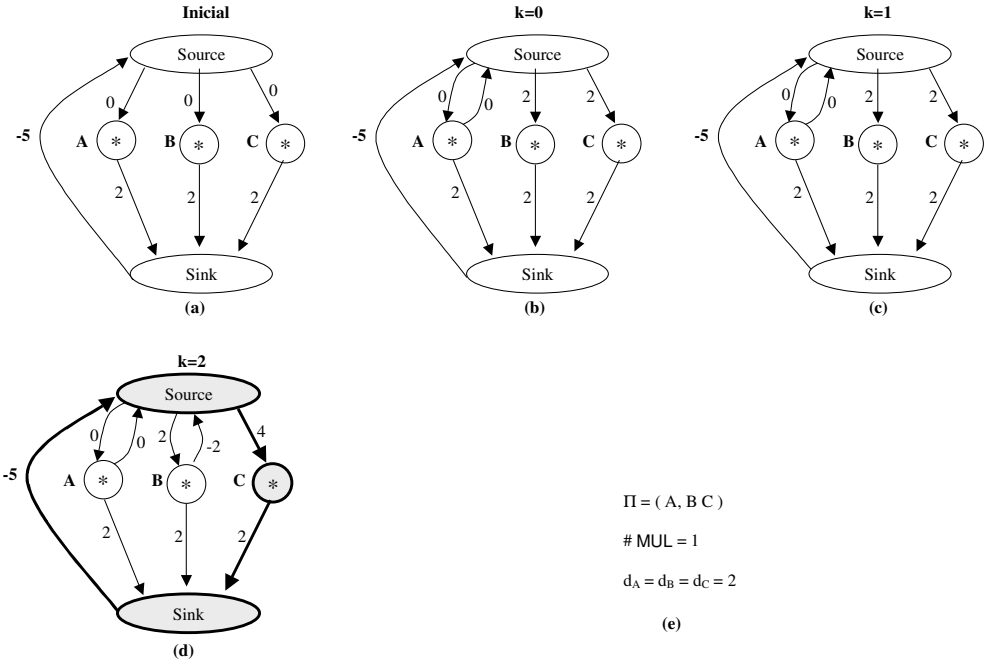

Figura 3.7: Refinando a modelagem para melhor refletir restrições de recursos.

Observe na Figura 3.7c que os pesos das arestas (Source,B) e (Source,C) são mantidos, já que $dr(A,s_1)=1$, logo, $k+ dr(A,s_1) = 2$. No estado s_2 (Figura 3.7d), com o recurso livre, a B é escalonada. O peso da aresta (Source,C) é atualizado para $k+dr(B,s_2) = 4$, resultando em infactibilidade.

Esse é um exemplo trivial, no qual somente uma operação é escalonada por vez devido à restrição de recursos de um único multiplicador. Considerando DFGs com grande número de vértices e com vários recursos, deve-se tomar cuidado para que operações não sejam postergadas além do necessário. Para evitar atrasos desnecessários, o atraso remanescente utilizado para determinar o peso da restrição a ser imposta a uma operação v_i é o menor dentre as operações em execução, num dado estado s_k, que ocupam os recursos do tipo $\tau(v_i)$, como formalizado a seguir.

Definição 3.2: Dados um estado s_k e um tipo de recurso t, seja um conjunto $P = U_k \cup \{v_j \mid \varphi(v_j) = k\}$. O mínimo atraso remanescente no estado s_k, das operações que ocupam recursos do tipo t, denotado por $dr_{min}(t,s_k)$, é $dr_{min}(t,s_k) = \min\{dr(v_i,s_k) \mid v_i \in P \wedge \tau(v_i) = t\}$.

O peso de uma aresta, representando uma restrição de atraso mínimo imposta a uma operação $v_i \in V$ não escalonada por falta de recursos livres do tipo $\tau(v_i)$, num estado s_k, é $k+dr_{min}(\tau(v_i),s_k)$. Essa restrição representa o tempo mínimo no qual um recurso estará livre.

A utilização de restrições de tempo para modelar o impacto de outras restrições é utilizada em [Mesman97], que unifica em sua modelagem todas as restrições consideradas. Porém, no caso das restrições de recursos, a restrição de tempo é relacionada às operações que disputam o recurso. Em nossa abordagem as restrições adicionais têm origem no vértice fonte para que não se estabeleçam novas relações de precedência entre as operações.

3.4.3 Discussão sobre a Verificação de Infactibilidade

Há situações em que, mesmo sem considerar as restrições de recursos, é possível detectar a infactibilidade do Problema 2.2. A Figura 3.8 mostra um exemplo para o qual não existe escalonamento factível.

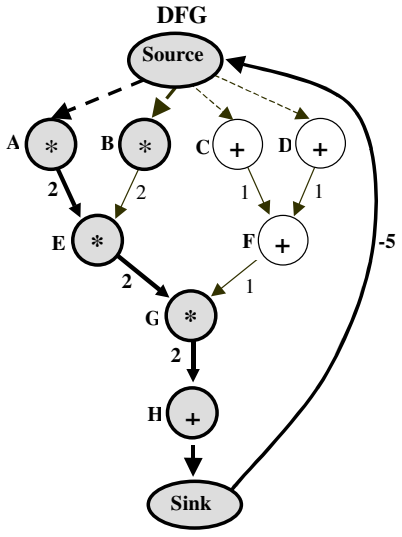

Figura 3.8: Exemplo de inconsistência.

Note que o comprimento do caminho mais longo entre os pólos, representado pela soma dos pesos das arestas assinaladas, supera, em módulo, a restrição de latência imposta, representada pela aresta (Sink,Source). Observe que as arestas assinaladas e a restrição de latência determinam a formação de um ciclo no DFG. Note que a distância percorrida nesse ciclo tem valor positivo. Num DFG um ciclo cujo comprimento é positivo significa que a soma dos atrasos mínimos é maior do que a soma dos atrasos máximos pertencentes ao ciclo.

A Figura 3.9 mostra um outro exemplo. Note que o comprimento do ciclo em destaque é positivo. Analisando as restrições de tempo, verifica-se que a operação B deveria ser escalonada, no mínimo, dois ciclos de relógio depois de A e, no máximo, um ciclo depois de Source. Como nenhuma solução pode obedecer a ambas restrições, qualquer escalonamento resulta em infactibilidade.

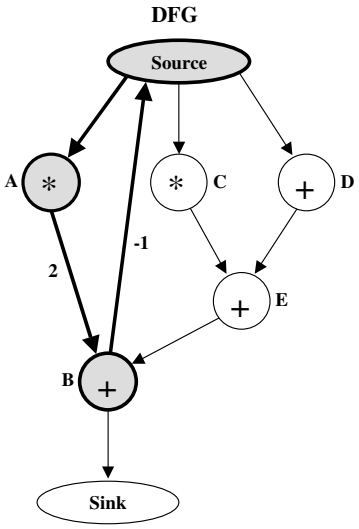

Figura 3.9: Outro exemplo de inconsistência

Os exemplos das Figuras 3.8 e 3.9 ilustram situações que levam à noção de *inconsistência* [DeMicheli94], como formalizado abaixo.

Definição 3.3: Um grafo orientado e ponderado G(V,E,W) é *inconsistente* se, e somente se, existe um ciclo $c \in V$ tal que $l(c) > 0$.

Como, durante a construção de uma solução, restrições de tempo são inseridas no DFG para capturar decisões de escalonamento, tais restrições alteram o comprimento dos caminhos no DFG e podem resultar na formação de ciclos com comprimento positivo. Assim, é natural que a verificação de inconsistência do DFG seja a base da verificação de factibilidade do Analisador de Restrições.

Ao auxiliar na resolução do Problema 2.2 (sob restrições de tempo e de recursos), o Analisador verifica a factibilidade em duas situações distintas:

- *Verificação estática*: consiste em verificar a infactibilidade do Problema 2.3 (recursos ilimitados), utilizando um algoritmo exato, antes de iniciar a resolução do Problema 2.2 (recursos limitados).

- *Verificação dinâmica*: consiste em verificar se uma solução em construção é infactível.

Note que o Analisador não pode garantir a factibilidade do Problema 2.2 a priori, pois a abordagem de resolução adotada não é exata. Por outro lado, o Analisador garante a detecção de infactibilidade para Problema 2.2 caso seja detectado para o Problema 2.3.

Nesta dissertação, são propostas duas alternativas para verificação de infactibilidade. A primeira baseia-se na aplicação de um algoritmo exato tanto para verificação estática quanto dinâmica. A segunda sugere um algoritmo aproximado para a verificação dinâmica, a fim de acelerar o processo de construção de soluções. O algoritmo proposto baseia-se na atualização incremental do comprimento do caminho mais longo entre os pólos passando por um dado vértice v. As duas alternativas são explicadas a seguir.

3.4.4 Verificação de Infactibilidade Utilizando o Algoritmo de Bellman-Ford

O algoritmo de Bellman-Ford como descrito em [Cormen90] é utilizado originalmente para solucionar o problema clássico de *caminho mais curto* a partir de um único vértice-fonte (“Single Source Shortest Path”), assim enunciado: Dado um grafo G(V, E, W) e dado um vértice arbitrário $s \in V$, determinar para todo vértice $v \in V$ o caminho $s \rightarrow v$ tal que a distância percorrida entre s e v seja mínima.

Para suportar o problema de escalonamento, o algoritmo é adaptado para resolver o problema de *caminho mais longo*, que se resume em: Dado um grafo orientado G(V, E, W) e dado um vértice arbitrário $s \in V$, determinar para todo vértice $v \in V$ o caminho $s \rightarrow v$ tal que a distância percorrida entre s e v seja máxima. Neste caso a inconsistência é caracterizada pela existência de um ciclo com comprimento positivo.

No Algoritmo 3.1, o procedimento Bellman-Ford(G(V, E, W), s) retorna valor verdadeiro se o caminho mais longo foi encontrado e retorna valor falso se o grafo G(V, E, W)) é inconsistente. Neste procedimento, d[u] representa a distância entre os vértices s e u, p[u] representa o predecessor imediato do vértice u no caminho mais longo e w_{uv} representa o peso da aresta (u, v).

O algoritmo baseia-se na técnica de relaxação das arestas do grafo, refinando sucessivamente a estimativa do caminho mais longo.

O procedimento inicializa as distâncias invocando o procedimento InicializeCaminhos(V,s). Em seguida, cada aresta (u,v) do grafo é visitada e a distância d[v] é refinada em função da distância de seu predecessor u e do peso w_{uv} da aresta, invocando-se o procedimento Refine(u,v). Em seguida, o procedimento Bellman-Ford(G(V, E, W),s) verifica a consistência do problema observando para cada aresta (u,v), as distâncias d[u] e d[v] e o peso da aresta .

Quando o procedimento retorna o valor verdadeiro, cada vértice $v \in V$ tem sua distância máxima em relação a s corretamente anotada no atributo d. A seqüência de vértices do caminho mais longo pode ser determinada a partir de um vértice $v \in V$ em questão, visitando-se recursivamente o vértice anotado no atributo p até encontrar o vértice s, a partir do qual o caminho é iniciado.

```
InicializeCaminhos(V,s){
  para todo v ∈ V faça {
    d[v] = ∞ ;
    p[v] = nulo;
  }
   d[s] = 0;
}

Refine( u, v ) {
    se d[v] < d[u] + w_uv então {
       d[v] = d[u] + w_uv;
       p[v] = u;
    }
}

Bellman-Ford( G(V,E,W),s) {
  InicializeCaminho(V,s);
  para i de 1 até |V| -1 faça
     para cada aresta (u,v) ∈ E faça
       Refine(u,v);

  para cada aresta (u,v) ∈ E faça
     se d[v] < d[u] + w_uv
       retorne(falso);
  retorne(verdadeiro);
}
```

Algoritmo 3.1: Algoritmo de Bellman-Ford adaptado para determinação do caminho mais longo.

Esse algoritmo tem complexidade $O(V\,E)$. A utilização do Algoritmo de Bellman-Ford dá origem à primeira alternativa de verificação de infactibilidade, como mostra o Algoritmo 3.2, onde v_0 é o pólo fonte do DFG.

```
Infactível ( DFG(V, E, W) )
{
     θ = ⌉ Bellman-Ford( DFG (V, E, W), v_0);
     retorne θ;
}
```

Algoritmo 3.2: Verificação de infactibilidade usando o algoritmo de Bellman-Ford

3.4.5 Detecção de Infactibilidade Baseada em Atualização Incremental

A motivação para desenvolver ou utilizar um outro mecanismo para verificação da factibilidade é manter a eficiência desse processo sem aumentar em demasia a complexidade do Analisador. O método aqui proposto tem por objetivo diminuir a

necessidade de executar um algoritmo que determine o caminho mais longo sempre que uma operação for escalonada.

Lembre que, para modelar o fato de que uma operação v_i foi escalonada no estado atual s_k, arestas são inseridas no DFG (Seção 3.4.1). Recorde também que, para modelar o fato de que v_i não foi escalonada no estado atual devido à carência de recursos, o peso da aresta incidente em v_i é incrementado (Seção 3.4.2).

No primeiro cenário, o escalonamento de v_i pode aumentar o comprimento do caminho mais longo podendo causar a violação da restrição de latência máxima. No segundo cenário, o adiamento do escalonamento de v_i pode causar a violação do atraso máximo de v_i em relação a alguma outra operação.

Esses dois cenários são a base de um algoritmo para verificação dinâmica da infactibilidade, proposto a seguir. As Seções 3.4.5.1 e 3.4.5.2 mostram os requisitos de inicialização e atualização de distâncias que amparam o algoritmo de verificação, que é descrito na Seção 3.4.5.3.

3.4.5.1 Inicialização, Decomposição e Determinação do Caminho mais Longo

Lembre que, o caminho mais longo entre dois vértices num DFG representa o atraso mínimo entre as operações representadas por esses vértices. Portanto, o caminho mais longo entre os pólos representa o tempo total mínimo necessário para que todas as operações sejam escalonadas.

O algoritmo de verificação de infactibilidade a ser proposto é amparado pela noção de caminho mais longo entre os pólos passando por um vértice arbitrário $v_i \in V$, denotado por LPT[v_i] (“Longest Path Through v_i”).

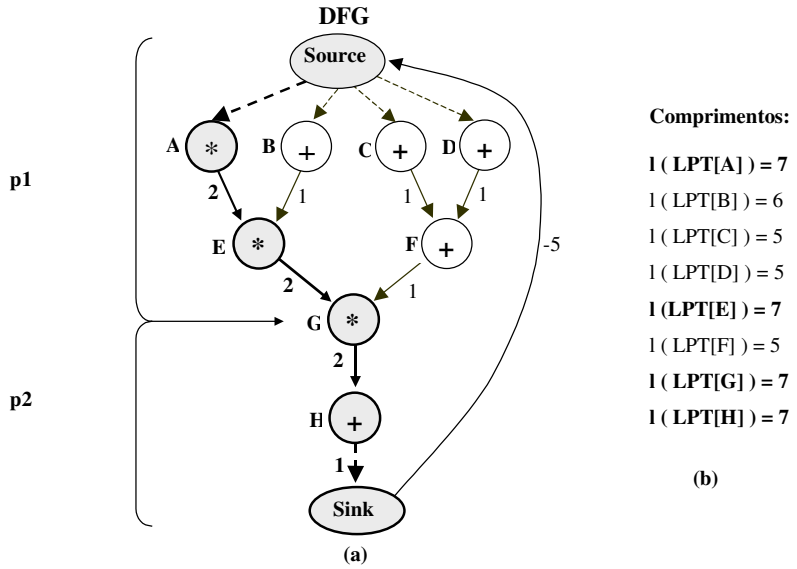

Figura 3.10: Caminho mais longo determinado em função de LPT[v_i].

Na Figura 3.10a, o caminho p = (Source, A, E, G, H, Sink) é o mais longo entre Source e Sink. Observe que p = LPT[G] pode ser decomposto em Source $\xrightarrow{p1}$ G e G $\xrightarrow{p2}$ Sink. Note na Figura 3.10b que, para um vértice arbitrário $v_i \in$ p, l(LPT[v_i]) = l(p), ou seja, o comprimento do caminho mais longo é obviamente o mesmo visto a partir de qualquer um de seus vértices.

Isto sugere que, durante o escalonamento, após a inserção de novas arestas, o valor LPT[v_i] pode ser utilizado para a determinação do comprimento do caminho mais longo.

As estimativas iniciais, para cada vértice $v_i \in$ V, do comprimento dos caminhos mais longos entre Source e v_i e v_i e Sink, podem ser determinadas utilizando o algoritmo de Bellman-Ford durante a verificação estática de factibilidade.

3.4.5.2 Atualização Incremental das Distâncias

Seja d[v_i] a distância percorrida no caminho mais longo entre Source e v_i, calculada pelo Algoritmo 3.1, quando da inicialização. Considerando que uma operação v_i está pronta somente se todos os seus predecessores foram escalonados e se todas as restrições de tempo impostas estão satisfeitas, ao se inserir arestas com peso maior que

$d[v_i]$, pode-se atualizar o comprimento do caminho mais longo entre Source e v_i invocando o Algoritmo 3.3.

```
AtualizeLP(v_i,w_0i)          // w_0i é o peso da aresta (Source,v_i)
{
 se (w_0i > d[v_i])
        d[v_i] = w_0i;
}
```

Algoritmo 3.3: Atualização da distância do caminho mais longo.

Considerando que todas as restrições de tempo inseridas após o escalonamento ser iniciado têm origem em Source, o caminho mais longo entre v_i e Sink não é modificado. Assim para manter atualizado o comprimento do caminho $LPT[v_i]$ pode-se usar o algoritmo que segue.

```
AtualizeLPT(v_i)
{
  Seja p o caminho mais longo entre v_i e Sink;
   l( LPT[v_i] ) = d[v_i] + l(p);
}
```

Algoritmo 3.4: Atualização de $LPT[v_i]$.

Os Algoritmos 3.3 e 3.4 são utilizados para manter atualizado o comprimento do caminho mais longo entre os pólos sem que seja necessária a execução de um algoritmo específico para determinação de tal caminho.

3.4.5.3 Verificação de Infactibilidade

Como visto anteriormente, a noção de inconsistência, que resulta em infactibilidade, é baseada na análise de ciclos. Um ciclo de comprimento positivo implica em inconsistência, como mostram os exemplos das Figuras 3.8 e 3.9 (Seção 3.4.3).

Revisando a Figura 3.8, nota-se que o caminho mais longo entre os vértices Source e Sink supera a restrição de latência, enquanto que a Figura 3.9 mostra a violação de uma restrição de atraso relativo entre operações. Ao contrário do algoritmo de Bellman-Ford, que é capaz de detectar essas duas situações simultaneamente, a técnica de atualização incremental aqui proposta as trata separadamente.

A primeira situação é tratada pelo Algoritmo 3.5 e a segunda, pelo Algoritmo 3.6, apresentados a seguir. Esses algoritmos são combinados no Algoritmo 3.7 para realizar a verificação dinâmica de infactibilidade, como mostrado no final desta Seção. Dado um vértice v_i ao qual foi imposta uma restrição de atraso mínimo, o Algoritmo 3.5 utiliza o comprimento do caminho mais longo através de v_i para verificar se a latência máxima foi excedida.

```
SatisfazRestriçãoLatência(v_i, w_0i)
{
  AtualizeLP(v_i,w_0i);
  AtualizeLPT(v_i);
  λ_max = |w_n0|  // peso da aresta (Sink,Source) em módulo

  Se l( LPT[v_i] ) ≤ λ_max
       retorne verdadeiro;
  Senão
     retorne falso;
}
```

Algoritmo 3.5 Verificação dinâmica da restrição de latência.

Ao se escalonar um estado, aplica-se o Algoritmo 3.5 apenas às operações que foram escalonadas nesse estado e àquelas a que foram impostas restrições de tempo por não haverem recursos disponíveis. Assim, somente as operações afetadas por decisões de escalonamento contribuem para o tempo computacional do Analisador.

Dado um vértice v_i escalonado em um estado s_k ou pronto para nele ser escalonado, o Algoritmo 3.6 visita todas as arestas dele emergentes, cujos pesos são negativos. Para cada aresta (v_i,v_q) visitada, a diferença entre os tempos de início de v_q e v_i é comparada com o atraso máximo especificado. O algoritmo retorna falso se pelo menos uma restrição de atraso máximo é violada.

```
SatisfazAtrasosMáximos (v_i, s_k)
{
      para cada sucessor v_q de v_i tal que w_iq  < 0 faça
            se ( (t_q – k) < w_iq )
                  retorne falso;
     retorne verdadeiro;
}
```

Algoritmo 3.6: Verificação de inconsistência analisando atrasos máximos.

Dado um vértice v_i, afetado por uma decisão de escalonamento, o Algoritmo 3.7 verifica se a decisão resulta em solução infactível.

```
Infactível(v_i, s_k)
{
   retorne ( ⌉( SatisfazRestriçãoLatência(v_i) ∧ SatisfazAtrasosMáximos(v_i)  ) );
}
```

Algoritmo 3.7: Verificação dinâmica de infactibilidade utilizando a técnica de atualização incremental.

3.4.6 Apreciação das Técnicas Propostas

O Algoritmo de Bellman-Ford é um algoritmo clássico, aplicável a qualquer topologia de grafo e com várias propriedades provadas matematicamente. Embora a utilização deste algoritmo torne o Analisador robusto, também aumenta sua complexidade computacional.

Por outro lado, a técnica baseada em atualização incremental resulta em menor esforço computacional. Entretanto, até o momento não há provas matemáticas para ampará-lo. A validação da técnica será feita experimentalmente.

3.4.7 A Técnica de Recuperação de Soluções

O Analisador de Restrições age anotando no DFG o efeito do escalonamento de cada operação (Seção 3.4.1), bem como o efeito de seu não escalonamento por falta de operadores disponíveis (Seção 3.4.2). Ademais, faz a verificação de factibilidade (Seção 3.4.3). Além disso, o Analisador tenta explorar a detecção de infactibilidade para melhorar a qualidade das soluções geradas, dando origem a uma técnica de recuperação de soluções, ilustrada pelos exemplos a seguir.

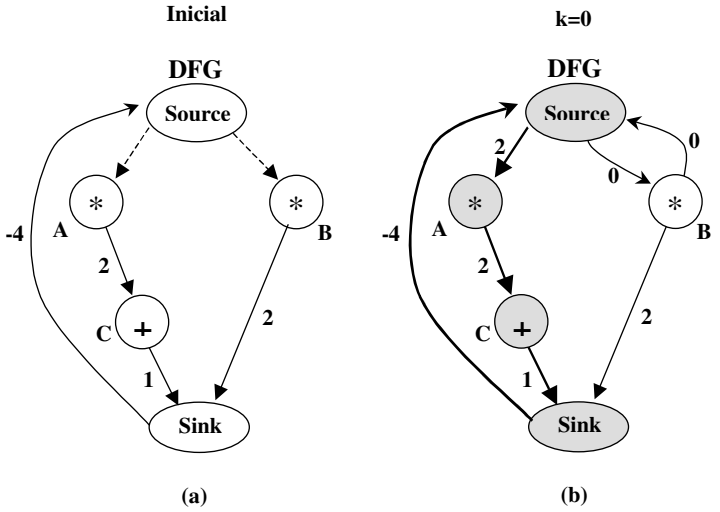

Figura 3.11: Detecção de infactibilidade pelo analisador de restrições.

A Figura 3.11 mostra um exemplo de comportamento do Analisador. Para este exemplo, assume-se um operador de cada tipo (uma ALU e um multiplicador). Assume-se também que a operação de soma tem um atraso de execução de 1 ciclo e que a multiplicação tem um atraso execução de 2 ciclos, conforme anotado nas arestas emergentes dessas operações. Observe ainda que a restrição de latência é $\lambda_{max} = 4$ e assuma que a codificação de prioridade Π é tal que B tem prioridade sobre A.

Na Figura 3.11a mostra-se o DFG antes do escalonamento. Note que as operações A e B estão prontas. Por força da prioridade Π, B é selecionada, como mostra a Figura 3.11b, e fixada no estado s_0, enquanto o não escalonamento de A é anotado. Ao verificar-se a factibilidade, obtém-se θ=falso, já que o comprimento de valor 5 do caminho mais longo supera a restrição de latência $\lambda_{max} = 4$, resultando num ciclo com comprimento positivo. A solução é, então, abortada.

Uma vez detectada infactibilidade, observa-se a última decisão de escalonamento efetuada na tentativa de revogá-la. Considera-se que mudanças numa decisão que causa infactibilidade podem vir a tornar a solução factível. No exemplo da Figura 3.11 a decisão a ser revogada seria a de escalonar a operação B ao invés de A no primeiro estado.

Na tentativa de recuperar a solução, o Analisador insere uma restrição de atraso mínimo (Source,B) com peso (t_B+1), para que a operação tenha início, pelo menos, um ciclo de relógio mais tarde. No processo de recuperação de uma solução, a mesma codificação de prioridade Π é utilizada. Esta idéia está ilustrada na Figura 3.12.

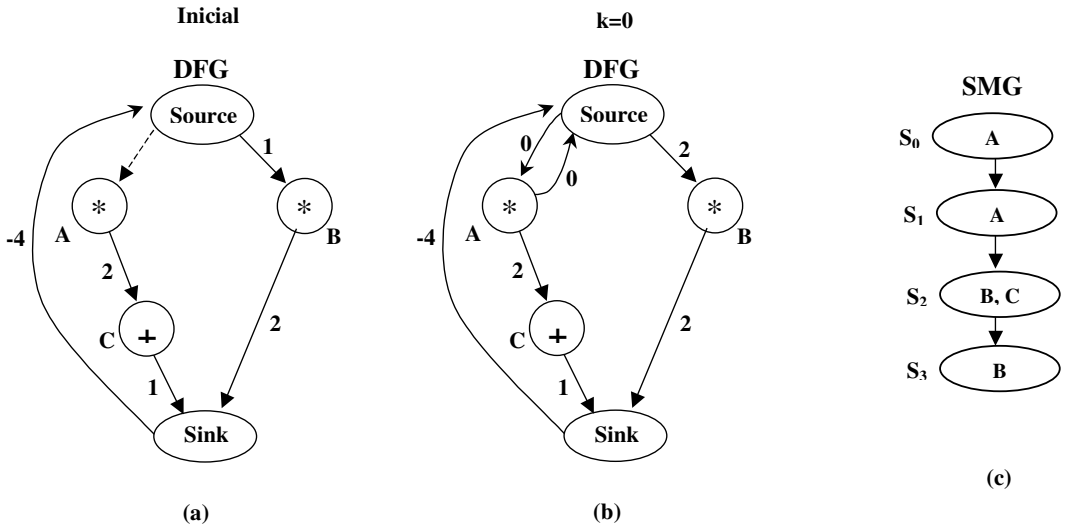

Figura 3.12: Restrição de tempo influenciando no resultado do escalonamento.

Na Figura 3.12a está o DFG antes do novo escalonamento ser iniciado. Note que, com a restrição de atraso mínimo atribuída para fins de recuperação à aresta (Source,B), B só estará pronta no segundo estado. Logo, A é escalonada no primeiro estado, como mostrado na Figura 3.12b. A Figura 3.12c traz o SMG resultante ao final do escalonamento. Note que B executa em paralelo com C no estado s_2, o que não ocorreria no exemplo da Figura 3.11. Interpretando as restrições de tempo, o Analisador pôde tornar a solução factível sem alterar a codificação de prioridade, premissa básica da abordagem explorativa adotada.

Em resumo, dada uma codificação de prioridade Π, ao detectar que uma solução resultaria infactível, o Analisador não se limita a abortar a sua construção. Ao contrário, o Analisador identifica a decisão de escalonamento que causaria infactibilidade e impõe restrições de tempo adicionais para impedir que aquela decisão seja novamente tomada. Com isso, a mesma codificação Π, que induziria infactibilidade, pode induzir uma solução factível.

Revisite a Figura 3.12b e note que o comprimento do caminho LPT[B] tem valor 4, que é igual à restrição de latência. Uma restrição de atraso mínimo (Source,B), tendo valor maior do que 2, tornaria a solução infactível. Tal observação sugere o estabelecimento de um tempo de início máximo para cada operação, ou seja, o quão tarde essa operação poderia ser escalonada sem resultar em infactibilidade. Esse valor é chamado de ALAP ("As Late as Possible") e é calculado através do Algoritmo 3.8, adaptado de [DeMicheli94].

```
ALAP (DFG(V, E, W), λmax )
{
  ALAP[vn] = λmax;
  marque vn;
  repita
  {
     selecione um vértice não marcado vi ∈ V com todos os seus sucessores marcados;
     ALAP[vi] = min { ALAP[vj] - wij | ∀ vj, (vi,vj) ∈ E };
     marque vi;
  } até que v0 esteja marcado;
}
```

Algoritmo 3.8: Cálculo do máximo tempo de início (ALAP).

Não faz sentido que um vértice v_i tenha atraso mínimo total em relação ao vértice fonte com valor maior que ALAP[v_i], pois isso tornaria o grafo inconsistente. O Analisador verifica essa condição antes de inserir restrições de atraso mínimo durante o processo de recuperação de uma solução, pois neste caso a tentativa de recuperação seria inócua, resultando em nova solução infactível.

No exemplo das Figuras 3.11 e 3.12 apenas uma restrição de tempo precisou ser adicionada para que a solução a princípio abortada se tornasse factível, porém existem situações não triviais que dificultam a recuperação da solução. Observe o exemplo da Figura 3.13 que assume como restrição de recurso apenas um multiplicador.

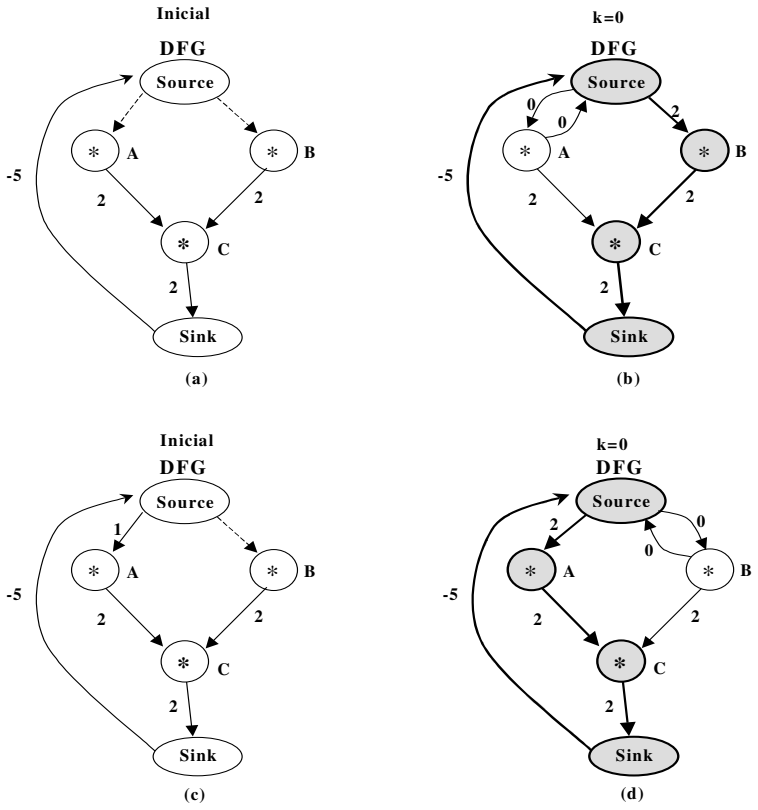

Figura 3.13: Falha na recuperação de soluções.

A Figura 3.13a mostra o DFG antes do escalonamento ser iniciado, estando as operações A e B prontas. A Figura 3.13b mostra que a operação A é escalonada e fixada no estado s_0, enquanto o não escalonamento de B é anotado. Neste estado é verificada infactibilidade, pois o ciclo (Source, B, C, Sink, Source) tem comprimento positivo.

Como no exemplo das Figuras 3.11 e 3.12, o Analisador insere uma restrição de tempo na intenção de modificar decisões de escalonamento. Observe na Figura 3.13c, um atraso mínimo (Source,A) de um ciclo anotado no DFG, que será novamente escalonado. Com somente a operação B pronta, ela é escalonada e fixada no estado s_0, e novamente é verificada infactibilidade, como mostra a Figura 3.13d.

O Analisador atuaria anotando uma restrição de tempo (Source,B) que não recuperaria a solução. Por isso, deve-se ter um parâmetro para definir o quanto insistir na recuperação de uma solução. O Analisador pressupõe um parâmetro T, que estabelece o número máximo de tentativas de recuperação, como será mostrado no Capítulo 4.

4 Implementação e Experimentos

Este capítulo apresenta os algoritmos implementados para a resolução dos Problemas 2.1 (escalonamento sob restrição de recursos) e 2.2 (escalonamento sob restrições de recurso e de tempo). As implementações seguem o paradigma utilizado no já mencionado Grupo de Trabalho OASIS, no âmbito do Projeto DESERT. Esse Projeto adota como plataforma-base um microcomputador PC utilizando o sistema operacional Linux, o ambiente de trabalho KDE [Kde03] e o ambiente de desenvolvimento KDevelop [Kdevelop03], usando a linguagem C++ [Stroustrup00].

Essencialmente, a implementação aqui descrita é resultado de duas contribuições principais:

- a extensão do algoritmo de escalonamento apresentado em [Azambuja01] para suportar restrições de tempo, e
- a agregação do Analisador de Restrições (Seção 3.4), estendendo a abordagem original (Seção 3.2) para modelar decisões de escalonamento e o impacto das restrições de recursos, verificar infactibilidade e, possivelmente, melhorar a qualidade das soluções através da técnica de recuperação (Seção 3.4.7).

4.1 Construção de Soluções

O processo de construção de uma solução consiste basicamente no escalonamento das operações de um DFG, mapeando-as para estados de um SMG. Na medida em que os estados são criados, as operações prontas e operações pendentes são mapeadas para esses estados. Os algoritmos a seguir apresentam com mais detalhes a construção de uma solução para o Problema 2.2, além dos processos de verificação de infactibilidade e recuperação de soluções.

Dado um estado s_k, o primeiro passo é selecionar uma operação no conjunto das operações prontas ou no conjunto das operações pendentes para escalonamento nesse estado. O Algoritmo 4.1 apresenta o procedimento SelecioneOperação(U_k, A_k, **a,** Π),

que escolhe uma operação em A_k ou U_k em função de sua prioridade na codificação Π e das restrições de recursos, retornando a operação selecionada.

```
SelecioneOperação (U_k, A_k, a, ∏) {
  se (U_k = ∅) então
  {
   escolha a operação v_i ∈ A_k com maior prioridade ∏ que satisfaz a restrição a;
   retire v_i de A_k;
  }
  senão
  {
   escolha a operação v_i ∈ U_k com maior prioridade ∏;
   retire v_i de U_k;
  }
  retorne(v_i);
}
```

Algoritmo 4.1: Seleção de uma operação para escalonamento.

Após ser selecionada, uma operação é escalonada no estado s_k através do procedimento EscaloneEstado(s_k, A_k, U_k, **a**, $\prod$, λ_{max}), apresentado no Algoritmo 4.2. Esse procedimento escalona operações no estado s_k enquanto operações puderem ser selecionadas pelo Algoritmo 4.1. As operações escalonadas em s_k têm seu tempo de início anotado e são adicionadas ao conjunto OP_k de operações escalonadas no estado s_k, além de serem fixadas neste estado invocando o procedimento Fixe(v_i,s_k), que consiste na adição de duas arestas representando um atraso exato, como explicado na Seção 3.4.1. Como as arestas inseridas representam uma decisão de escalonamento que pode vir a ser revogada, essas arestas são marcadas pelo Algoritmo 4.2 para distinguí-las das demais arestas durante o processo de recuperação de soluções (Seção 3.4.7). Em seguida, o Analisador de Restrições retorna o resultado da verificação de infactibilidade. Ao final, o procedimento EscaloneEstado(s_k, A_k, U_k, **a**, $\prod$, λ_{max}) retorna o par (OP_k,θ).

```
Fixe (v_i, s_k) {                          // captura da decisão de escalonamento
        AdicioneAresta (v_0,v_i) no DFG com w_0i = d(s_0, s_k)
        AdicioneAresta (v_i,v_0) no DFG com w_i0 = - d(s_0, s_k)
        marque (v_i,v_0);
        marque (v_0,v_i);
}

EscaloneEstado (s_k, A_k, U_k, a, ∏, λ_max ) {
        OP_k = ∅;
        θ = falso;
        v_i  = SelecioneOperação (U_k, A_k, a, ∏);
        Enquanto (  ( v_i ≠ nenhuma ) ∧ ( θ = falso )  ) faça
        {
                escalone v_i em s_k;
                t_i = k;
                OP_k = OP_k ∪ {v_i};
                Fixe(v_i, s_k);
                θ = Analise(A_k, OP_k, a, v_i, λ_max);
                v_i  = SelecioneOperação (U_k, A_k, a, ∏);
        }
        retorne(OP_k, θ);
}
```

Algoritmo 4.2: Escalonamento das operações num estado.

Fixar uma operação num estado significa anotar, na forma de restrições de tempo, o efeito do seu escalonamento. A uma operação não escalonada devido à carência de recursos também é imposta uma restrição de tempo adicional (Seção 3.4.2). A primeira dessas tarefas é realizada pelo já explicado procedimento Fixe(v_i,s_k). A segunda, é realizada pelo procedimento Analise(A_k, OP_k, a, v_i, λ_{max}), apresentado no Algoritmo 4.3.

Inicialmente, o Algoritmo 4.3 anota o impacto da restrição de recursos (Seção 3.4.2), adiando todas operação v_j que não dispõe de um recurso livre do tipo $\tau(v_j)$. As arestas inseridas para adiar operações são marcadas para que sejam distinguidas de outras arestas durante o processo de recuperação. Em seguida, invoca-se a verificação de infactibilidade, a qual pode ser implementada através de uma das duas versões propostas: Infactível(DFG(V, E, W)) ou Infactível(v_i, s_k), descritas nos Algoritmos 3.2 e 3.7, respectivamente. Caso seja detectada infactibilidade, uma restrição de tempo adicional é inserida no DFG para fins de recuperação da solução. Note que, como

explicado na Seção 3.4.7, o peso de uma restrição de atraso mínimo imposta a uma operação v_i tem valor máximo limitado por ALAP[v_i].

```
Analise (A_k, OP_k, a, v_i, λ_max) {
  Para todo v_j ∈ A_k faça          // captura do impacto da restrição de recursos
  {
       Se (não existe recurso livre do tipo τ(v_j) ) então
            AdicioneAresta(v_0,v_i) com w_0i = k + dr_min(τ(v_j),s_k);
            marque (v_0,v_i);
  }

  θ = Infactível ( ... );            //verificação dinâmica de infactibilidade

  Se (θ = verdadeiro) então
  {
       AdicioneAresta (v_0,v_i);
       w_0i = min { (t_i +1) , ALAP[v_i] } ;   // muda decisão de escalonamento

  }
  retorne θ;
}
```

Algoritmo 4.3: Modelagem e análise de restrições.

Uma vez escalonadas todas as operações possíveis de A_k em s_k e feitas a modelagem e análise das restrições, buscam-se operações para serem escalonadas no estado s_{k+1}. Uma operação está pronta para escalonamento quando todos os seus predecessores já foram escalonados (Definição 2.8) e quando todas as restrições de atraso mínimo impostas a ela estão satisfeitas. O Algoritmo 4.4 apresenta o procedimento SatisfazAtrasosMínimos(v_i,s_k) que verifica os atrasos mínimos em relação a um dado vértice v_i no estado s_k.

```
SatisfazAtrasosMínimos(v_i, s_k) {
       para cada predecessor v_q de v_i tal que w_qi ≥ 0 faça
          se ( (k – t_q) < w_qi )
               retorne falso;
        retorne verdadeiro;
}
```

Algoritmo 4.4: Verificação de atrasos mínimos.

O procedimento EncontreOperaçõesProntas(OP_k, s_k), apresentado no Algoritmo 4.5, é responsável por encontrar o maior número de operações que podem executar em

paralelo. Este procedimento também define quais operações sucessoras daquelas presentes em OP_k tornaram-se prontas para escalonamento no próximo estado.

```
EncontreOperaçõesProntas (OPk, sk) {
  P = ∅;
  Para cada operação vi ∈ OPk faça
  {
    Para cada sucessor vj de vi no DFG
      Se ( (todos os predecessores de vj foram escalonados) ∧
              (SatisfazAtrasosMínimos(vj,sk) )
             P = P ∪ {vj};
  }
retorne(P);
}
```

Algoritmo 4.5: Atualização do conjunto de operações disponíveis.

As operações multiciclo não completadas num estado s_k ficam pendentes no estado s_{k+1}. No Algoritmo 4.6, cada operação não completada em s_k é inserida num conjunto P, que é retornado pelo procedimento EncontreOperaçõesPendentes(OP_k, s_k).

```
EncontreOperaçõesPendentes (OPk, sk) {
  P = ∅;
  Para cada operação vi ∈ OPk faça
  {
       Se ( dr(vi,sk) > 1 ) então
             P = P ∪ {vj};
  }
retorne(P);
}
```

Algoritmo 4.6: Atualização do conjunto de operações pendentes.

O SMG resultante do escalonamento é construído pelo Algoritmo 4.7 a partir das restrições de recursos **a**, da codificação de prioridade Π e da restrição de latência λ_{max}. Inicialmente, o estado s_0 é criado e o conjunto A_0 calculado. A cada iteração do laço um estado é escalonado invocando o Algoritmo 4.2 (que chama o Algoritmo 4.3 para a análise de restrições de tempo e verificação de infactibilidade).

Em caso de infactibilidade, o Algoritmo 4.7 retorna o valor $\lambda=\infty$, indicando que a solução é infactível. Em caso contrário, a estimativa parcial da latência λ é atualizada e

busca-se por operações pendentes para o próximo estado (Algoritmo 4.6) e por novas operações prontas (Algoritmo 4.5).

Enquanto houver operações pendentes ou prontas, um novo estado s_{k+1} é criado e será o próximo a ser escalonado. Os conjuntos U_{k+1} e A_{k+1} são atualizados e, na próxima iteração, o estado s_{k+1} é escalonado. O conjunto A_{k+1} é a união das operações que se tornaram prontas por força de predecessores escalonados em s_k (P) e das operações que nele não puderam ser escalonadas, remanescentes em A_k. O conjunto U_{k+1} é a união das operações multiciclo que iniciaram sua execução em s_k (Q) e das operações já pendentes em s_k, mas que nele não terminaram sua execução. Quando todas as operações forem escalonadas, o Algoritmo 4.7 retorna a latência λ.

```
Escalonador (a, ∏, λmax) {
        λ = 0;
        crie estado inicial s0 no SMG;
        A0 = { vi | vo é o único predecessor de vi no DFG};
        U0 = ∅;
        próximo = s0;
        Enquanto ( próximo ≠ nenhum ) faça
        {
                sk = próximo;
                (OPk, θ)  = EscaloneEstado (sk, Ak, Uk, a, ∏, λmax);
                Se (θ = verdadeiro) então
                {
                  λ = ∞;
                  retorne λ;
                }
                λ = λ +1;
                P = EncontreOperaçõesProntas (OPk, sk);
                Q = EncontreOperaçõesPendentes(OPk, sk);
                Se ( (P ≠ ∅) ∨ (Q ≠∅) ) então
                {
                        Crie um novo estado sk+1 no SMG;
                        Ak+1 = Ak ∪ P;
                        Uk+1 = Uk ∪ Q;
                        próximo ← sk+1;
                }
                senão
                        próximo ← nenhum;
        }
        retorne (λ);
}
```

Algoritmo 4.7: Escalonamento e formação do SMG.

Considerando que o valor de retorno do Algoritmo 4.7 seja λ tal que $\lambda > \lambda_{max}$, a solução pesquisada é infactível, logo o DFG será escalonado novamente utilizando a mesma codificação de prioridade. A diferença entre o escalonamento que resultou em infactibilidade e a nova tentativa reside no fato de que restrições de tempo foram adicionadas ao DFG de forma que possam modificar decisões de escalonamento (vide Algoritmo 4.3), como explicado na Seção 3.4.7.

O Construtor tem como parâmetros de entrada um DFG, um vetor **a** de restrição de recursos, uma codificação de prioridade Π, uma restrição de latência λ_{max} e um número máximo de tentativas para recuperar a solução T (Seção 3.4.7), conforme mostra o Algoritmo 4.8.

O Construtor retorna o custo da solução pesquisada. Para o caso de uma solução que não pôde ser recuperada esse valor é ∞. Em caso contrário, o custo é tal que $\lambda \leq \lambda_{max}$. O custo da solução para a codificação Π é avaliado com a chamada do procedimento Escalonador(**a**, Π, λ_{max}) do Algoritmo 4.7 até que este retorne um valor $\lambda \leq \lambda_{max}$ ou até que o número de tentativas de recuperação exceda T.

Após cada tentativa mal sucedida, restrições de tempo adicionais marcadas pelo Analisador (vide Algoritmo 4.3), são removidas do DFG. Os estados e transições do SMG são removidos e sua construção é reiniciada.

```
Construtor (DFG(V, E, W), a, λmax, T, Π) {
        num = 0;
        λ = Escalonador(a, Π, λmax);
        Enquanto ( (λ > λmax) ∧ (num < T) ) faça
        {
           remova todas arestas marcadas;
           S = ∅;
           T = ∅;
           num = num +1;
           λ = Escalonador(a, Π, λmax);
        }
        retorne (λ);
}
```

Algoritmo 4.8: Algoritmo do Construtor de soluções.

4.2 Exploração de Soluções Alternativas

Como mostrado na Seção 3.2, a exploração de soluções alternativas é uma tarefa realizada pela interação entre os blocos Explorador e Construtor. O Algoritmo 4.9 formaliza a construção de N soluções para o Problema 2.2. Inicialmente é feita a verificação estática da infactibilidade (Seção 3.4.3), usando o Algoritmo 3.2. Em seguida são determinados os valores ALAP para todos os vértices. A cada iteração do laço, o Explorador gera uma codificação de prioridade, que será utilizada pelo Construtor para construir uma solução.

```
ExploraSoluçõesAlternativas (N, T, DFG(V, E , W), a, λmax)
{
 Se ( ⌉Infactível( DFG(V, E, W) ) ) então  // verificação estática de infactibilidade
 {
  ALAP( DFG(V, E, W), λmax);

  Para i de 1 até N faça {
    Π = Explorador(V);
    λ = Construtor(DFG(V, E, W), a, Π, λmax, T);
  }
 }
}
```

Algoritmo 4.9: Exploração de soluções alternativas.

Como o bloco Explorador não é abordado neste trabalho, não se apresenta aqui um algoritmo que demonstre seu funcionamento. Atualmente, o Explorador limita-se a gerar aleatoriamente codificações de prioridade.

4.3 Resultados Experimentais

O objetivo dos experimentos a seguir relatados é mostrar a atuação do Analisador (veja Figura 3.3) no processo de exploração automática de soluções para o problema de escalonamento. Para avaliar o impacto da utilização do Analisador, os resultados experimentais são comparados com os resultados encontrados com a utilização do escalonador apresentado em [Azambuja01], que utiliza a mesma abordagem explorativa, porém sem considerar restrições de tempo. Para os experimentos, foram

utilizados para os testes, exemplos extraídos da literatura de Síntese de Alto Nível, descritos a seguir.

O exemplo *diffeq* é baseado em um algoritmo utilizado para resolver equações diferenciais encontrado em [DeMicheli94]. O exemplo *wdelf* é um algoritmo que implementa um filtro de onda digital de quinta ordem, encontrado em [DeWilde85]. O outro exemplo utilizado é o *fdct* que calcula a "fast discrete consine transform" encontrado em [Mallon90]. Todos esses exemplos também foram utilizados em [Azambuja01]. A Tabela 4.1 apresenta as características dos DFGs resultantes desses exemplos em termos de número de vértices, arestas e dos tipos de operações que os compõem.

Tabela 4.1: Resumo das características dos DFGs utilizados como casos de teste (benchmarks).

Exemplo	Vértices	Arestas	Operações
diffeq	13	16	2 adições, 2 sub., 1 comp. e 6 mult.
fdct	44	68	13 adições, 13 sub., e 16 mult..
wdelf	36	66	25 adições, 1 sub., e 8 mult.

O número de vértices e arestas apresentados na Tabela 4.1 inclui os vértices pólos dos DFG resultantes, bem como as entradas primárias (arestas com origem no vértice fonte) e saídas (arestas com destino ao vértice sumidouro).

Para todos os experimentos considerou-se que operações mapeadas para multiplicadores levam dois ciclos de relógio para serem executadas, enquanto operações mapeadas para uma Unidade Lógico-Aritmética (adições, subtrações e comparações) necessitam de apenas um ciclo.

O impacto da utilização do Analisador é observado analisando-se o espaço de soluções e comparando os resultados encontrados com os que seriam obtidos sem o Analisador como mostrado mais detalhadamente nas seções seguintes.

4.3.1 O Impacto da Latência Máxima na Factibilidade

Uma vez definida a latência máxima (λ_{max}) para que uma solução seja considerada satisfatória, não faz sentido que qualquer solução com custo superior seja construída. Assim o Analisador de Restrições interpreta as restrições de forma a detectar o quanto

antes se uma dada solução será factível ou não. Os gráficos a seguir mostram curvas que representam a quantidade de soluções abortadas.

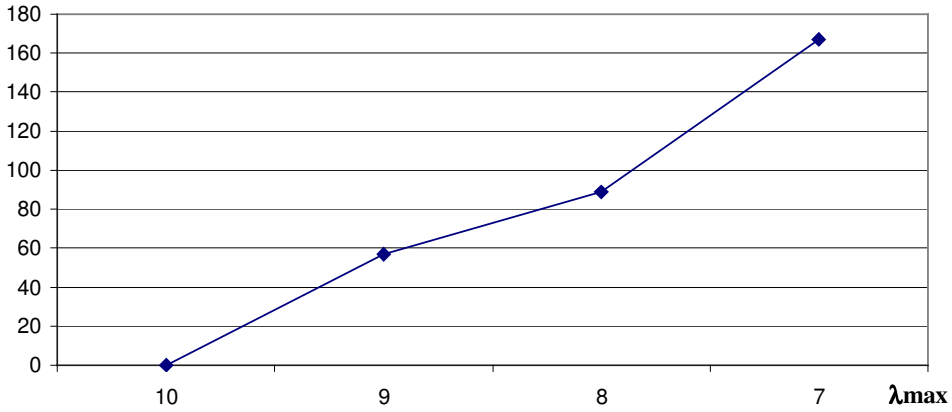

Gráfico 4.1: Número de soluções infactíveis para diferentes restrições de latência (diffeq).

O Gráfico 4.1 mostra o número de soluções abortadas para diferentes restrições de latência. No experimento, foram geradas 200 codificações de prioridade para o exemplo diffeq, sob uma restrição de recursos de dois operadores de cada tipo (2 Multiplicadores e 2 ALU's).

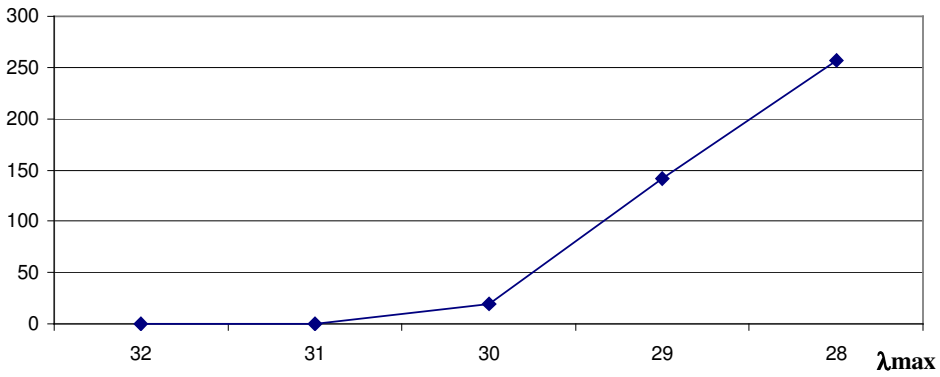

Gráfico 4.2: Número de soluções infactíveis para diferentes restrições de latência (wdelf).

Para o exemplo wdelf, foram geradas 500 codificações de prioridades para cada experimento. Para este exemplo a restrição de recursos considerada foi de um operador de cada tipo.

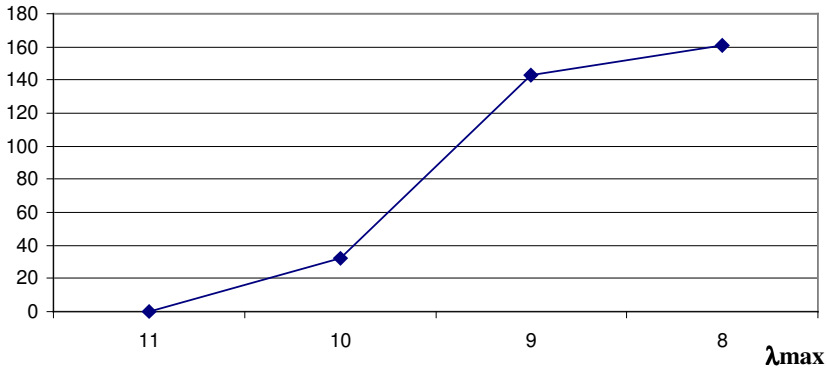

Gráfico 4.3: Número de soluções infactíveis para diferentes restrições de latência (fdct).

Para o exemplo fdct, 200 codificações de prioridades foram geradas para cada experimento. Assumiu-se como restrição de recursos 8 multiplicadores e 4 ALUs.

Observe em todos os exemplos que, para um dado número de soluções exploradas, o número de soluções factíveis diminui com a severidade da restrição de latência. Dessa forma menos tempo é gasto na construção de soluções que resultariam infactíveis, pois sua construção é abortada tão logo seja detectada infactibilidade.

4.3.2 O Impacto na Qualidade Média das Soluções

Um resultado importante diz respeito à capacidade do Analisador de recuperar soluções infactíveis. Como já explicado na Seção 3.4.7, se o Analisador detecta que a solução será infactível (com custo $\lambda > \lambda_{max}$), sua construção é abortada, são adicionadas restrições de tempo que influenciam a tomada de decisões de escalonamento e a solução é reconstruída a partir da mesma codificação de prioridade, esperando produzir uma solução factível ($\lambda \leq \lambda_{max}$). Uma solução assim construída é chamada de *solução factível recuperada*, enquanto uma solução com custo inicial $\lambda \leq \lambda_{max}$, obtida sem a adição de restrições, é chamada de *solução factível natural*. Assim, para um mesmo conjunto de codificações de prioridade, o Analisador aumenta a probabilidade de se obter mais cedo uma solução satisfatória.

Retirando soluções de baixa qualidade do espaço de soluções, e adicionando as soluções recuperadas, este apresenta uma melhora no que diz respeito à qualidade média das soluções que o compõem. As tabelas a seguir apresentam o custo médio das soluções

factíveis obtidas para os três exemplos utilizados em dois cenários distintos: *com* e *sem* o Analisador.

Tabela 4.2: Custo médio das soluções para o exemplo diffeq.

λ_{max}	Sem Analisador	Com Analisador
10	8,46	8,46
9	8,45	8,05
8	8,64	7,83
7	8,55	7

Tabela 4.3: Custo médio das soluções para o exemplo wdelf.

λ_{max}	Sem Analisador	Com Analisador
32	28,87	28,87
31	28,92	28,92
30	29,35	28,89
29	28,90	28,38
28	28,89	28

Tabela 4.4: Custo médio das soluções para o exemplo fdct.

λ_{max}	Sem Analisador	Com Analisador
11	9,79	9,79
10	9,57	9,3
9	9,78	8,67
8	9,61	8

Note que, com o uso do Analisador, o custo médio das soluções encontradas melhora com a severidade da restrição de latência. Em particular, quando a latência ótima é utilizada como restrição (7, 28 e 8 respectivamente para os três exemplos [Heijligers95]), nenhuma solução sub-ótima é construída quando o Analisador é utilizado.

Combinando os resultados dos Gráficos 4.1, 4.2 e 4.3, e das Tabelas 4.2, 4.3 e 4.4, conclui-se que, dado um número pré-fixado de codificações de prioridade, ao se aumentar a severidade da restrição de latência, embora diminua o número de soluções factíveis, as soluções remanescentes no espaço de soluções são de melhor qualidade. Essa melhora deve-se em parte à não construção de soluções insatisfatórias e em parte à recuperação de soluções infactíveis. Os Gráficos 4.4, 4.5 e 4.6 a seguir ilustram o percentual de recuperação de soluções para os experimentos realizados.

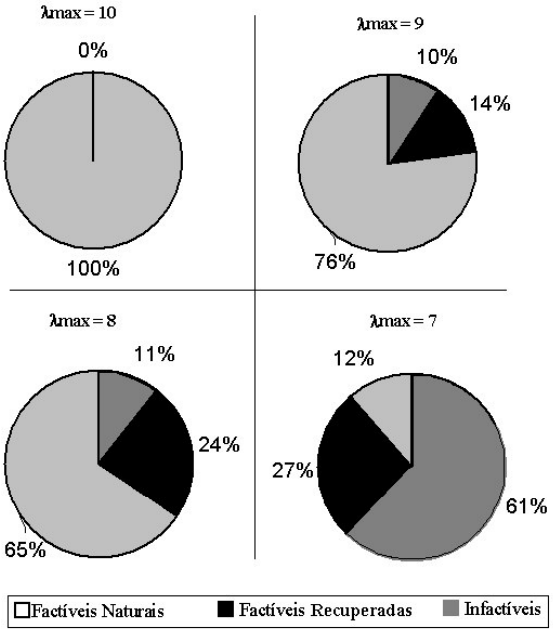

Gráfico 4.4: Recuperação de soluções melhorando a qualidade média do espaço de soluções (diffeq).

O Gráfico 4.4 apresenta setores correspondentes ao número de soluções abortadas, factíveis recuperadas e factíveis naturais. Observe que, para a restrição $\lambda_{max} = 10$, nenhuma solução foi abortada, já que essa foi a maior latência observada no espaço de busca. Para as restrições $\lambda_{max} = 9$ e $\lambda_{max} = 8$, a proporção de soluções recuperadas é maior do que o de soluções abortadas. Para a restrição $\lambda_{max} = 7$ o percentual de soluções factíveis recuperadas é maior do que o das soluções naturalmente factíveis, porém é bem menor do que o das soluções abortadas.

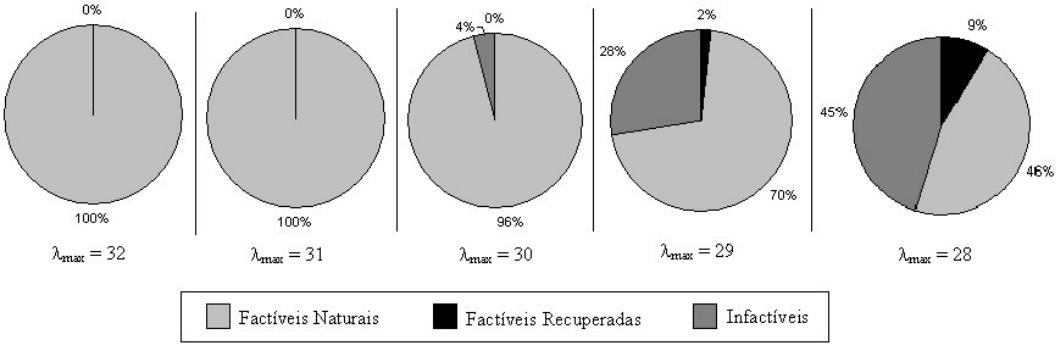

Gráfico 4.5: Recuperação de soluções melhorando a qualidade média do espaço de soluções (wdelf).

Para o exemplo wdelf (Gráfico 4.5) com restrição de latência $\lambda_{max} = 32$ e $\lambda_{max} = 31$ nenhuma solução foi abortada. Observe que o percentual de soluções recuperadas é bem menor do que no exemplo anterior. Isso se deve a características específicas do grafo e

ao fato da restrição de recursos não possibilitar muitas opções de paralelismo. Note que com restrição $\lambda_{max} = 30$ nenhuma solução é recuperada. Já com $\lambda_{max} = 29$ e $\lambda_{max} = 28$ observa-se a ação do Analisador recuperando soluções.

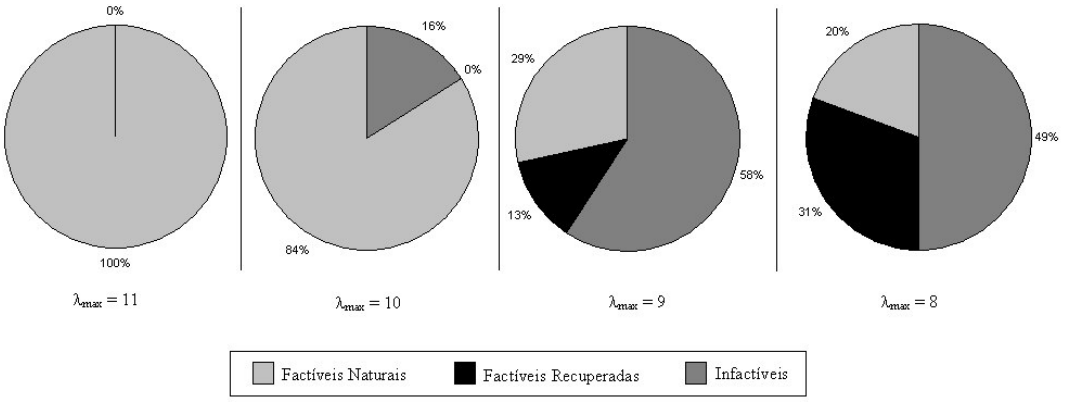

Gráfico 4.6: Recuperação de soluções melhorando a qualidade média do espaço de soluções (fdct).

Em testes com o exemplo fdct o mesmo panorama foi encontrado. Para $\lambda_{max} = 11$, maior latência observada, nenhuma solução foi abortada. Para $\lambda_{max} = 10$, observa-se soluções abortadas mas nenhuma recuperada. Com $\lambda_{max} = 9$ e $\lambda_{max} = 8$ percebe-se um aumento significativo das soluções recuperadas (Gráfico 4.6).

Os gráficos mostrados acima reforçam a noção de que, na medida em que a restrição de latência torna-se mais severa, o Analisador atua com maior freqüência. Ora, é justamente devido à severidade de restrições de tempo que um simples escalonador pode não ser capaz de encontrar uma solução factível. Portanto, o Analisador é útil exatamente nos casos em que um tal escalonador precisa de auxílio.

4.3.3 O Impacto na Convergência em Direção a Melhores Soluções

O fato de que a qualidade das soluções geradas melhora com o aumento da severidade da restrição de latência sugere uma técnica para melhorar a qualidade das soluções a serem geradas, valendo-se da qualidade das soluções já construídas, como descrito a seguir.

Em [Azambuja01] é feita uma observação importante sobre a abordagem utilizada: o processo de exploração do espaço de soluções por si só não garante uma convergência

das soluções encontradas em direção à latência ótima, como ilustra o Gráfico 4.7. Os custos nele mostrados foram obtidos para o exemplo diffeq, assumindo uma restrição de recursos de 2 multiplicadores e 2 ALUs e um conjunto de 200 codificações de prioridade geradas aleatoriamente.

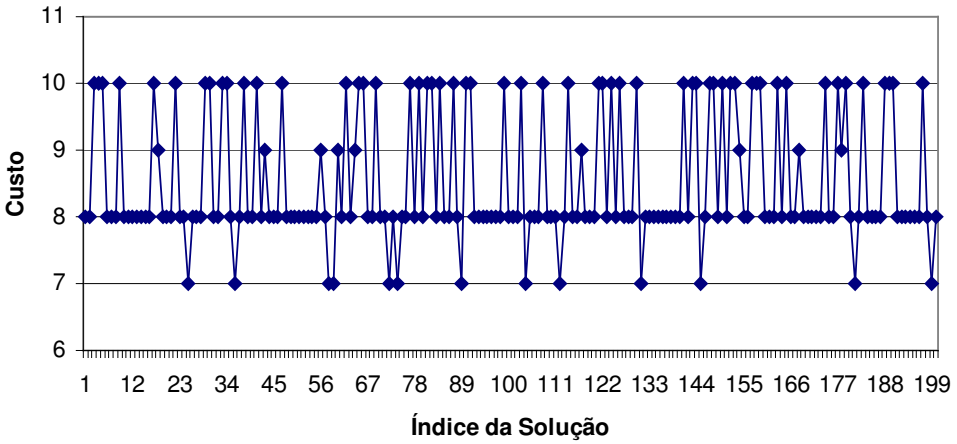

Gráfico 4.7: Distribuição dos custos sem utilização do Analisador para o exemplo diffeq.

Para obter tal convergência, sugere-se que sejam feitas melhorias no Explorador, como a utilização de meta-heurísticas ou algoritmos evolucionários. Entretanto, nenhuma melhoria é sugerida para o construtor, até porque na abordagem adotada é vedada a utilização de heurísticas no construtor para não limitar a exploração de melhores soluções.

Por outro lado, os resultados das Tabelas 4.2, 4.3 e 4.4 sugerem que é possível melhorar a qualidade das soluções sem limitar a exploração de melhores soluções, utilizando a menor latência até então observada, ou seja, usando uma *estimativa de latência reduzida* como restrição de latência imposta à próxima solução a ser gerada, conforme formalizado abaixo.

Definição 4.1 - Seja λ_{max} a restrição de latência do Problema 2.2. Assuma que uma solução de custo λ tenha sido construída. A *estimativa de latência reduzida* λ'_{max} é dada por:

- $\lambda'_{max} = \lambda$ se $\lambda < \lambda_{max}$ e
- $\lambda'_{max} = \lambda_{max}$ em caso contrário.

Note que esta técnica preserva soluções de custo inferior e, portanto, não limita a exploração de melhores soluções, o que está em acordo com a abordagem descrita na Seção 3.2.

A exploração de soluções alternativas como apresentada no Algoritmo 4.9, pode ser adaptada para que seja orientada a buscar pela solução de menor custo utilizando a Definição 4.1, conforme apresentado no Algoritmo 4.10.

```
ExploraSoluçõesAlternativas (N, T, DFG(V, E , W), a, λmax)
{
 Se ( ⌉Infactível( DFG(V, E, W) ) ) então
 {
  ALAP( DFG(V, E, W), λmax);
  λ'max = λmax;

  Para i de 1 até N faça {
    Π = Explorador(V);
    λ = Construtor(DFG(V, E, W), a, Π, λ'max, T);

     Se (λ < λ'max) então
          λ'max = λ;   // Definição 4.1
   }
 }
}
```

Algoritmo 4.10: Exploração de Soluções Orientada à Melhor Solução.

Para mostrar o impacto do Algoritmo 4.10, o experimento a que se refere o Gráfico 4.9 foi repetido. Para exatamente as mesmas codificações de prioridade e a mesma restrição de recursos, sob a restrição de tempo $\lambda_{max} = 10$, os resultados obtidos pelos Algoritmos 4.9 e 4.10 estão mostrados nos Gráficos 4.8a e 4.8b, respectivamente, para o exemplo diffeq, utilizando como restrição de recursos dois multiplicadores e duas ALUs.

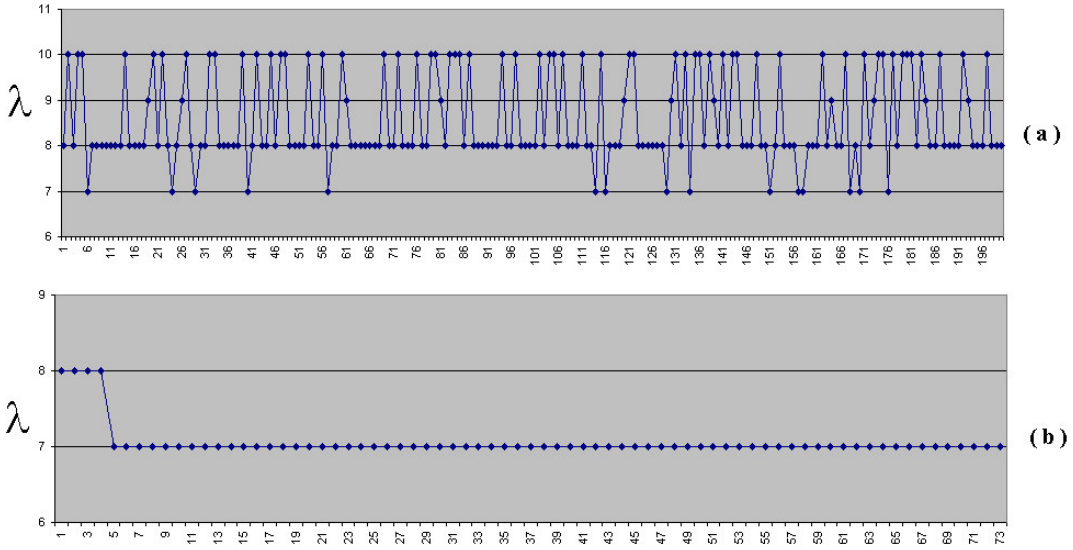

Gráfico 4.8 - Impacto do Algoritmo 4.8 na distribuição de custos para o exemplo diffeq.

É importante ressaltar que para distribuição de custos do Gráfico 4.8, somente as soluções factíveis foram apresentadas. É notória a melhoria de qualidade no espaço de soluções, devida à estimativa de latência reduzida.

O Gráfico 4.9 mostra três distribuições de custos para o exemplo diffeq. Note que, para diferentes restrições de latência iniciais, diferentes curvas de distribuição foram geradas. Na curva $\lambda_{max} = 10$, a latência ótima foi atingida na solução de índice 4, enquanto nas curvas $\lambda_{max} = 9$ e $\lambda_{max} = 8$ a latência ótima foi atingida nas soluções de índice 9 e 11, respectivamente.

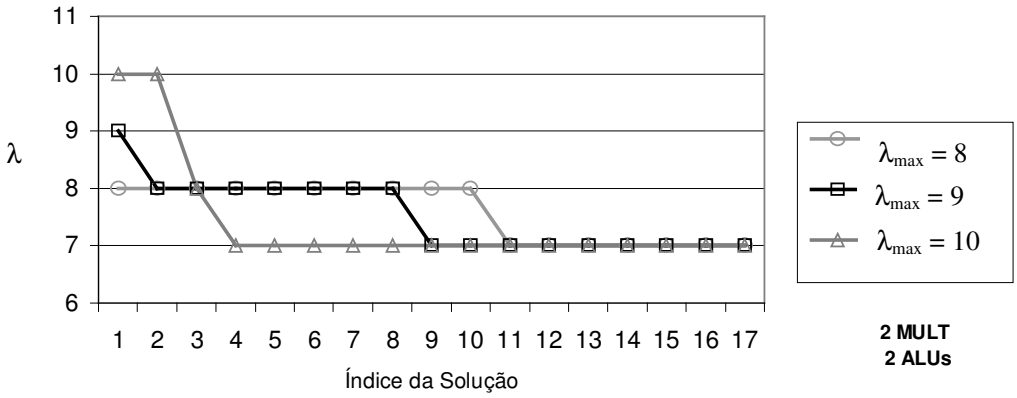

Gráfico 4.9: Convergência na busca de soluções de boa qualidade (diffeq).

Os Gráficos 4.10 e 4.11 a seguir, ilustram distribuições de custos a partir de diferentes restrições de latência iniciais para os exemplos wdelf e fdct, respectivamente. Note que o mesmo panorama é encontrado, ou seja, a convergência é atingida ainda nas primeiras soluções pesquisadas.

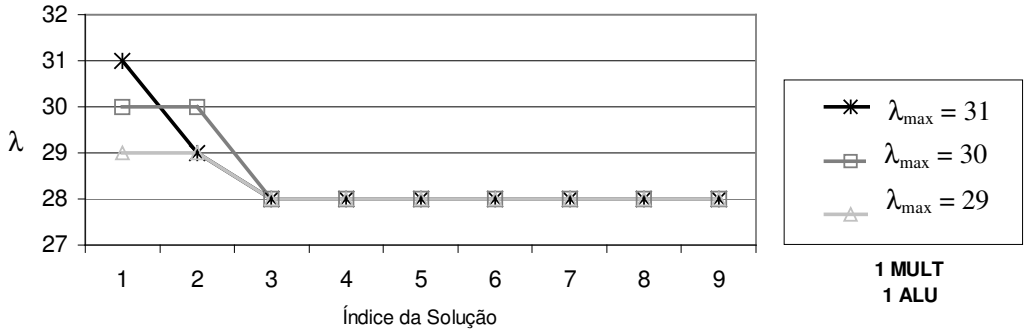

Gráfico 4.10: Convergência na busca de soluções de boa qualidade (wdelf).

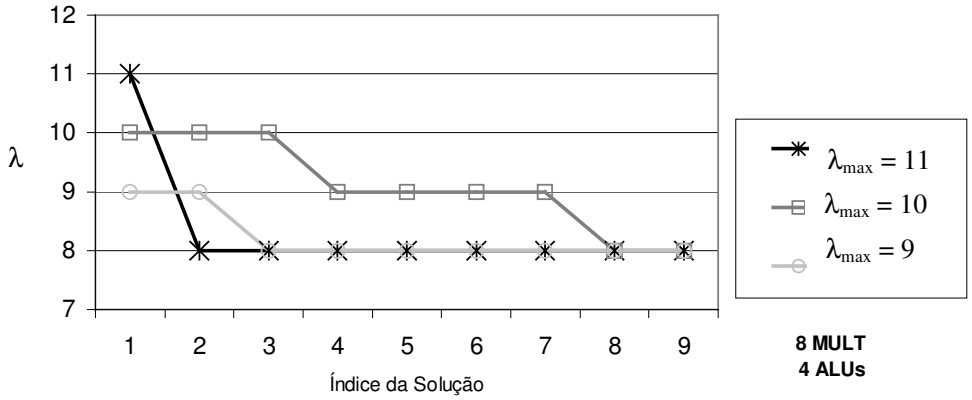

Gráfico 4.11: Convergência na busca de soluções de boa qualidade (fdct).

Nos Gráficos 4.9, 4.10 e 4.11 não se percebe nenhuma relação entre a restrição de latência inicial e o quão cedo a solução ótima foi atingida. Isto pode ser explicado pelo fato de o Explorador gerar as codificações de prioridade aleatoriamente. Porém, em todos os casos a convergência foi verificada. Portanto, com a utilização da estimativa de latência reduzida, o Analisador se mostra menos sensível à aleatoriedade das codificações de prioridade, garantindo a convergência em direção a melhores soluções.

4.3.4 O Impacto na Probabilidade de Atingir a Latência Ótima

O fato de o Analisador buscar uma convergência em direção a melhores soluções pode induzir a construção de uma solução ótima explorando um número menor de soluções comparado à abordagem original, que não leva em consideração restrições de tempo. Os experimentos realizados para avaliar a probabilidade de se atingir a latência ótima foram realizados da seguinte maneira.

Para cada exemplo, foram geradas várias seqüências de codificações de prioridade. Cada seqüência foi gerada aleatoriamente a partir de uma semente distinta. Cada seqüência gerada foi aplicada ao Escalonador sem a interferência do Analisador, produzindo um conjunto de soluções associado ao Escalonador. Em seguida, exatamente a mesma seqüência de codificações foi aplicada ao Escalonador, agora guiado pelo Analisador, gerando um outro conjunto de soluções associado ao Analisador. Finalmente, calculou-se a estimativa de probabilidade P(n) de se encontrar uma solução factível em até n soluções alternativas exploradas, tanto para o Escalonador quanto para o Analisador.

Os Gráficos 4.12, 4.13 e 4.14 mostram as probabilidades estimadas. Diferentes restrições de recursos foram utilizadas. Para o exemplo diffeq, 2 multiplicadores e 2 ALUs; para o exemplo fdct, 8 multiplicadores e 4 ALUs; e para o exemplo wdelf, 1 multiplicador e 1 ALU. Os valores utilizados como restrição de latência foram 7, 8 e 28 para diffeq, fdct e wdelf, respectivamente.

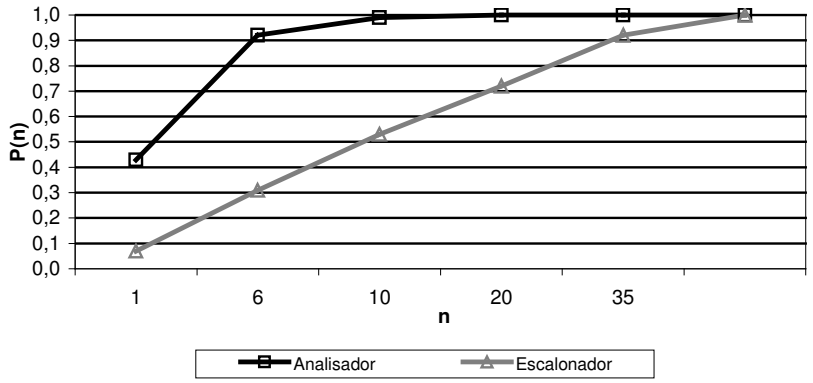

Gráfico 4.12: Estimativas de probabilidade para o exemplo diffeq.

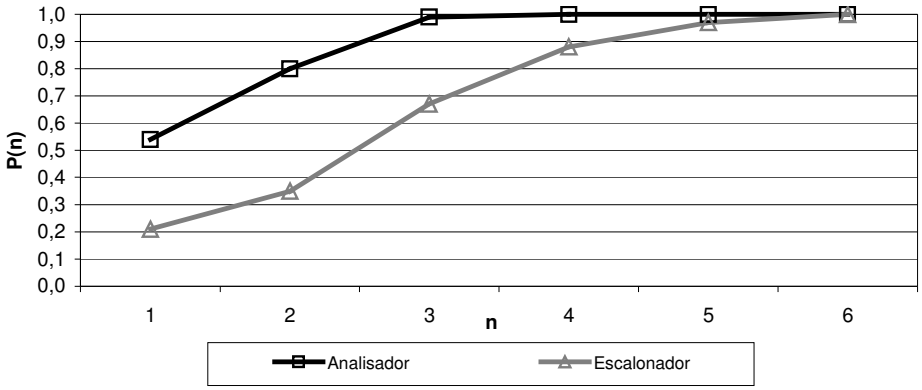

Gráfico 4.13: Estimativas de probabilidade para o exemplo fdct.

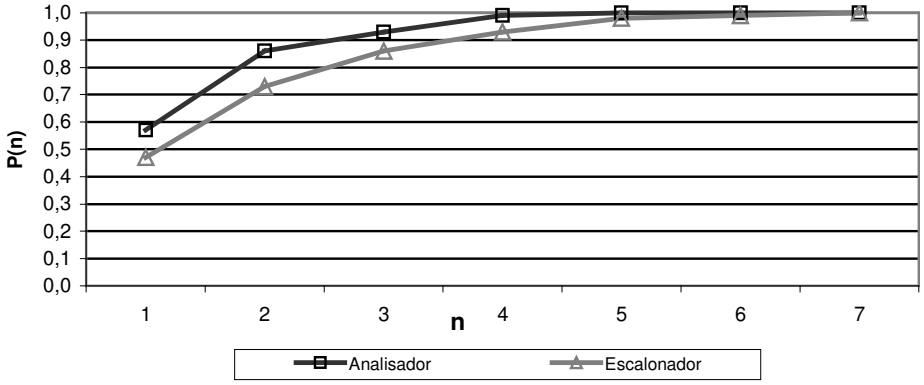

Gráfico 4.14: Estimativas de probabilidade para o exemplo wdelf.

Para o exemplo diffeq, note que a probabilidade de encontrar uma solução factível em 6 soluções pesquisadas é 0,9 com o Analisador e somente 0,3 para o Escalonador. Para o exemplo fdct, em 3 soluções pesquisadas, o Analisador garantiu factibilidade, enquanto que o Escalonador garantiu probabilidade de 0,7. Para o exemplo wdelf, o impacto do Analisador é menor: em 3 soluções pesquisadas a probabilidade aumentou de 0,86 para 0,93.

Revisando os Gráficos 4.4, 4.5 e 4.6, observe que nos exemplos diffeq e fdct a porcentagem de soluções recuperadas é maior do que no exemplo wdelf (27%, 31% e 9% para diffeq, fdct e wdelf, respectivamente). Os Gráficos 4.12, 4.13 e 4.14 mostram que a diferença entre a probabilidade estimada do Analisador e do Escalonador, é maior exatamente nos exemplos diffeq e fdct. Portanto, o impacto na probabilidade de se atingir a latência ótima parece ser o resultado da eficiência da técnica de recuperação descrita na Seção 3.4.7.

4.3.5 O Impacto no Tempo Médio para Atingir a Latência Ótima

Um outro experimento realizado a fim de avaliar o impacto da utilização do Analisador, diz respeito ao tempo médio para se atingir a latência ótima. Uma vez que, ao se agregar o Analisador, a complexidade do processo de construção de soluções é aumentada, é preciso avaliar se essa complexidade adicional é compensada pelo tempo de busca.

Para este experimento, o tempo médio para construir 100 soluções factíveis é medido com e sem o Analisador.

Foram utilizadas três versões distintas do Analisador. As versões diferem pelo mecanismo de verificação dinâmica de infactibilidade:

- Versão 1: O Algoritmo 3.2 (baseado no Algoritmo de Bellman-Ford) é aplicado após cada operação ser escalonada, como mostra o Algoritmo 4.2. Esta versão será denotada por BFO.
- Versão 2: O Algoritmo 3.2 é aplicado adaptando-se o Algoritmo 4.2 para invocar o Analisador somente ao final do escalonamento de cada estado. Esta versão será denotada por BFE (note que esta é uma tentativa de reduzir a complexidade do Analisador).
- Versão 3: O Algoritmo 3.7 é aplicado após cada operação ser escalonada, como mostra o Algoritmo 4.2. Esta versão será denotada por AIO.

A respeito da versão BFE, é importante ressaltar que, pelo fato de verificar infactibilidade apenas ao final de cada estado, o processo de recuperação de soluções pode ser prejudicado, já que várias decisões de escalonamento podem ser tomadas num mesmo estado. Assim, a chance de revogar a decisão exata que causou infactibilidade é menor do que nas versões BFO e AIO. A utilização das várias versões é justificada pelo fato de que elas agregam, ao escalonador original, complexidades diferentes.

A exemplo do experimento comentado na seção anterior, várias seqüências de codificações de prioridade foram geradas e aplicadas às diferentes versões. Porém, ao invés de se calcular estimativas de probabilidade, calculou-se estimativas de tempo. Essas estimativas consideram o número T de tentativas de recuperar uma solução.

O Gráfico 4.15 mostra as estimativas de tempo para as versões do Analisador. A curva relativa ao Escalonador refere-se ao uso do Escalonador, sem a interferência do Analisador. Essa curva apresenta valor constante, já que o Escalonador não efetua a recuperação de soluções.

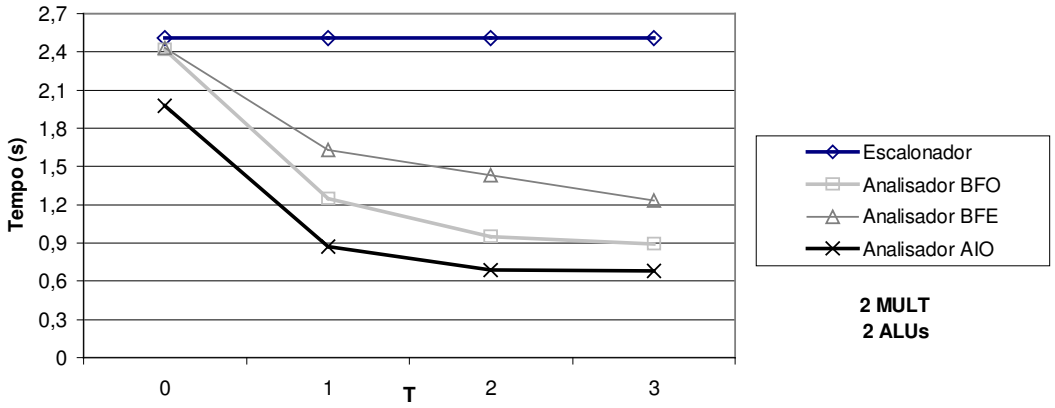

Gráfico 4.15: Estimativas de tempo para o exemplo diffeq.

Observe no Gráfico 4.15 que todas as versões do Analisador apresentaram tempos menores do que o Escalonador. Na medida em que o número de tentativas aumenta, o tempo até encontrar uma solução factível diminui. Note, no Gráfico 4.16, que o mesmo panorama foi encontrado para o exemplo fdct.

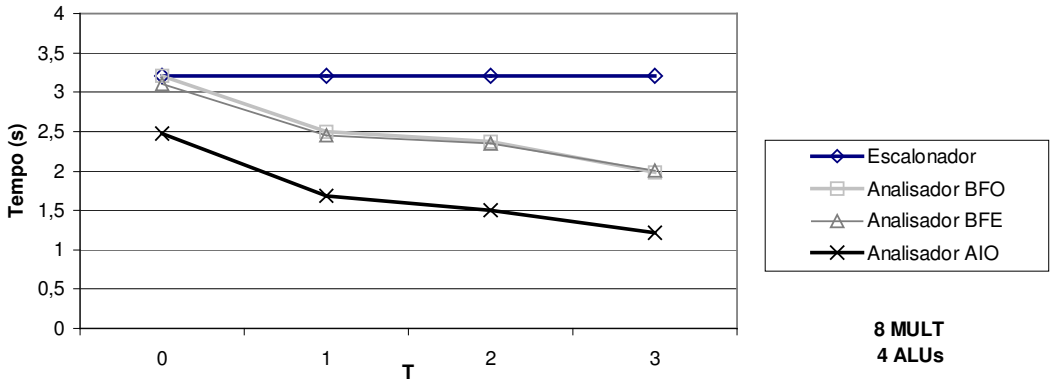

Gráfico 4.16: Estimativas de tempo para o exemplo fdct.

No Gráfico 4.17, onde são mostradas as estimativas de tempo para o exemplo wdelf, uma situação diferente é observada. Somente na versão AIO o Analisador atinge tempos menores do que o Escalonador.

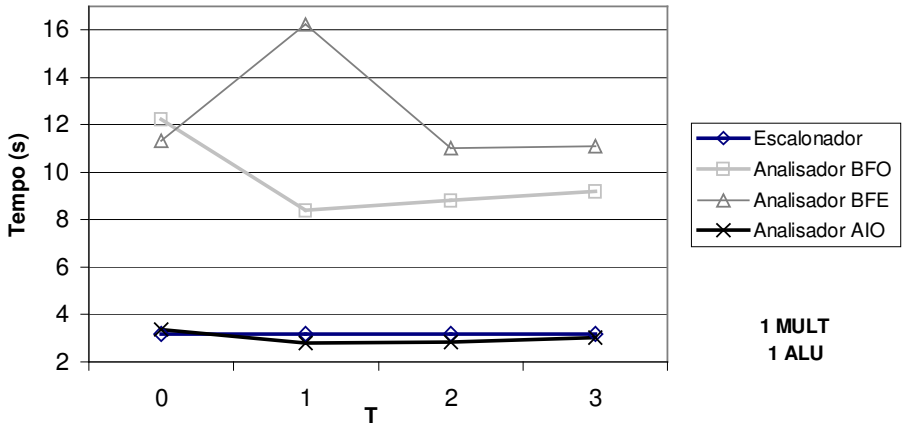

Gráfico 4.17: Estimativas de tempo para o exemplo wdelf.

Note que, no Gráfico 4.17, as duas versões do Analisador que utilizam o Algoritmo 3.2 apresentam estimativas com valores bem maiores do que o Escalonador, enquanto que a versão que utiliza o Algoritmo 3.7, apresenta estimativas mais próximas às do Escalonador.

Percebe-se também neste experimento, que existe uma relação entre as melhorias introduzidas pelo uso do Analisador, e a porcentagem de soluções recuperadas. Exatamente nos casos em que mais soluções foram recuperadas, o Analisador apresentou mais eficiência no que diz respeito ao tempo de execução. Para os casos em que a porcentagem de soluções recuperadas é baixa, as vantagens do Analisador se restringem a melhorar a qualidade média das soluções encontradas.

Algumas limitações da abordagem e sugestões de melhorias serão apresentadas e discutidas no Capítulo 5.

5 Conclusões e Perspectivas de Trabalhos Futuros

Esta dissertação apresentou um trabalho de pesquisa realizado no contexto da Síntese de Alto Nível, tendo como objeto principal de estudo o problema de escalonamento sob restrições de recursos e de tempo. As principais contribuições desta pesquisa em relação ao problema atacado e em relação à abordagem utilizada são:

- A aplicação da modelagem de restrições de tempo proposta em [DeMicheli94] à abordagem de Síntese proposta em [Santos98], tornando esta apta a resolver o problema de escalonamento sob restrições de recursos e de tempo.

- A extensão do escalonador apresentado em [Azambuja01] para suportar a modelagem de restrições de tempo.

- A proposta de um Analisador de Restrições que interage com o Escalonador através da modelagem de decisões de escalonamento.

- A proposta e implementação de técnicas para verificação estática e dinâmica da infactibilidade do problema abordado. Nas verificações estática e dinâmica, propôs-se a utilização de um algoritmo clássico exato.

- A proposta e implementação de um novo algoritmo para verificação dinâmica da infactibilidade do problema.

- A proposta e implementação de uma técnica para recuperação de soluções infactíveis.

Com base nos resultados apresentados, conclui-se que os objetivos de detectar infactibilidade na construção soluções e melhorar o espaço de soluções foram alcançados (Seções 4.3.1 e 4.3.2). Além disso, foi observada convergência na exploração de soluções em direção à melhor solução encontrada (Seção 4.3.3). Também foi verificado um impacto sobre a probabilidade de se encontrar soluções factíveis

(Seção 4.3.4) e no tempo médio para que uma solução factível seja encontrada (Seção 4.3.5).

Os resultados experimentais indicam que o Analisador de Restrições aumenta a probabilidade de encontrar-se uma solução factível. Entretanto, tal aumento de probabilidade poderia ser mera vantagem aparente, quando o custo agregado pelo Analisador resulta em tempo médio de execução superior ao do escalonador puro. Ora, os resultados mostram que o fator determinante para que o Analisador efetivamente contribua para a diminuição do tempo de busca por uma solução factível é a escolha de um algoritmo de verificação dinâmica de infactibilidade com baixa complexidade.

Embora o uso de um algoritmo exato não seja adequado para verificação dinâmica, sua utilização para verificação estática é praticamente mandatória: como o algoritmo é invocado uma única vez, o custo que agrega é marginal frente à vantagem de uma maior eficácia na detecção a priori da infactibilidade do problema.

Outra conclusão importante é que o custo adicional agregado pelo mecanismo de verificação dinâmica só é compensado se associado a um mecanismo de recuperação de soluções.

Embora algumas hipóteses simplificadoras tenham sido assumidas para viabilizar a execução do trabalho de pesquisa no tempo disponível, a modelagem e as técnicas de análise propostas foram concebidas para não perder sua generalidade quando removidas as hipóteses assumidas. Outros trabalhos já desenvolvidos sob a mesma abordagem propuseram técnicas para tratar descrições comportamentais mais genéricas, como as construções condicionais [Azambuja02] e os laços de iteração [Ferrari02]. Essas técnicas podem ser combinadas com as aqui propostas em futuros trabalhos de pesquisa, através de pequenas adaptações.

Algumas limitações da abordagem estão sendo removidas por trabalhos correlatos em andamento. O Explorador, que atualmente se limita a gerar codificações aleatoriamente, está sendo estendido. Nele está sendo embutido um algoritmo de simulação de têmpera (“Simulated Annealing”) [Amorim04] para melhorar a exploração de soluções

alternativas. O Construtor está sendo estendido para suportar restrições impostas por um número pré-fixado de barramentos [Oliveira04].

Para trabalhos futuros, sugere-se uma investigação adicional do impacto do parâmetro T do Algoritmo 4.8, que define o número de tentativas de recuperar uma solução. O valor desse parâmetro é fundamental para o equilíbrio entre o tempo computacional e a qualidade da solução obtida. Portanto, seu impacto merece ser avaliado através de experimentação com um maior número de “benchmarks”.

Uma outra sugestão, envolvendo outras áreas de pesquisa, é extensão e adaptação da modelagem aqui apresentada para possibilitar a um compilador a geração de software embutido sob restrições de tempo real.

6 Referências Bibliográficas

[Aiken95] AIKEN, A.; NICOLAU A.; "Resource-Constrained Software Pipelining". IEEE Transactions on Parallel and Distributed System, vol. 6, nº 12, Dezembro 1995.

[Amorim04] AMORIM, XENIA K.; "Aplicação de Algoritmo de Busca na Otimização de Sistemas Digitais" – Trabalho de Conclusão de Curso. Universidade Federal de Santa Catarina.

[Azambuja01] AZAMBUJA, ROGÉRIO X.; "Escalonamento e Otimização de Registradores para Grafos de Fluxo de Dados" – Trabalho Individual. Universidade Federal de Santa Catarina, 2001.

[Azambuja02] AZAMBUJA, ROGÉRIO X.; "Escalonamento e Alocação de Registradores Sob Execução Condicional" –Dissertação de Mestrado. Universidade Federal de Santa Catarina, 2002.

[Cormen90] CORMEN, THOMAS H. Et. al; "Introduction to Algorithms". McGraw Hill, 1990.

[DeMicheli94] DE MICHELI, GIOVANNI; Synthesis and Optimization os Digital Circuits, Mc Graw-Hill, 1994.

[DeWilde85] DEWILDE et al., "Parallel and Pipelined VLSI Implementation os Signal Processing Algorithms", in S. Y. Kung, H. J. Whitehouse and T. Kailath, VLSI and ModernSignal Processing, Prentice Hall, 1985.

[Ferrari02] FERRARI, DIONE J; "Aplicação de Loop Pipelining e Loop Unroling à Síntese de Alto Nível" –Dissertação de Mestrado. Universidade Federal de Santa Catarina, 2002.

[Gajski87] GAJSKI, D.; "Silicon Compilers", Addison-Wesley, Reading, MA, 1987.

[Heijligers95] HEIJLIGERS, M. J. M.; CLUITMANS, L. J. M.; JESS, J. A. G.; "High Level Synthesis Schedulingand Allocation using Genetic Algorithms", Proceedings of the Asia and South Pacific Design Automation Conference, pp 61-66, 1995.

[Kde03] KDE – The K Desktop Environment, disponível em www.kde.org, 2003

[Kdevelop03] Kdevelop, disponível em www.kdevelop.org, 2003

[Liao&Wong83] LIAO, Y.; WONG, C.; "An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints", IEEE Transactions on CAD/ICAS, 1983.

[Mallon90] MALLON, D. J.; DENYER, P. B., "A New Aproach to Pipeline Optimization", Proc. EDAC´90m, 1990.

[Mesman97] MESMAN, BART et. al; "Constrint Analysis for DSP Code Generation", X International Symposium on System Synthesis, 1997.

[Oliveira04] OLIVEIRA, VALTER M.; "Escalonamento e Otimização sob Restrições de Barramentos" – Dissertação de Mestrado. Universidade Federal de Santa Catarina.

[Santos98] SANTOS, LUIZ C. V. DOS; "Exploiting instruction-level parallelism: a constructive approach", Eindhoven University of Technology, PhD. Thesis, 1998.

[Stroustrup00] STROUSTRUP, BJARNE; "A Linguagem de Programação C++", Bookman, Porto Alegre, 2000.

[Timmer93] TIMMER, ADWIN H. A.H., JESS, J.A.G.; "Execution Interval Analysis under Resource Constraints", IEEE International Conference on Computer-Aided Design, pp 454--459, Santa Clara, Novembro 1993. IEEE Computer Society Press.

[Timmer95] TIMMER, ADWIN H. A.H., JESS, J.A.G.; "Exact Scheduling Strategies based on Bipartite Graph Matching", Proceedings of the European Design and Test Conference ED&TC (EDAC--ETC--EuroASIC), Paris-France, March, 1995.

[Wolf00] W. Wolf; "Computers as Components", MK Publishers, 2000.