APLICAÇÃO DE LOOP PIPELINING E LOOP UNROLLING À SÍNTESE DE ALTO NÍVEL

Dione J. Ferrari[*], *Luiz Cláudio V. dos Santos*[+], *Rogério X. de Azambuja*[°] *e Felipe Vieira Klein*[#]
Departamento de Informática e Estatística
UFSC – universidade Federal de Santa Catarina
Florianópolis, Santa Catarina / Brasil
[*]dione@inf.ufsc.br [+]santos@inf.ufsc.br [°]xavier@inf.ufsc.br [#]fvklein@inf.ufsc.br

ABSTRACT

The loop execution consumes a big portion of the application time execution. In this loops, instructions may be executed several times and in differents iterations. The time execution otimization of this structures using loop pipeline techniques, can produce significant results.

This paper presents one approach for otimization of this structures. It is based on Loop Pipelining and Loop Unrolling techniques applied in High-Level Synthesis. We present experimetal results using classical models of the high-level synthesis literature and comparisons between these results and others reported in literature.

The results are promising since we find data introduction interval (dii) better than others autors.We also find some damages, discuted in this papper.

RESUMO

Grande parte do tempo de execução de aplicações é gasto com execução de laços, onde as instruções internas a este laço devem ser executadas inúmeras vezes. Dessa forma, a otimização do tempo de execução dessas estruturas, através de técnicas de paralelização, pode produzir resultados significativos.

Este artigo apresenta uma abordagem para a otimização dessas estruturas. Nossa abordagem baseia-se na aplicação de técnicas de paralelização de laços ("Loop Pipelining" e "Loop Unrolling") à Síntese de Alto Nível. São apresentados os resultados de experimentos utilizando modelos clássicos da Síntese de Alto Nível e também comparações destes com outros resultados reportados por outros autores da literatura.

Os resultados se mostraram promissores uma vez que as comparações mostraram que encontramos melhores intervalos de introdução de dados (dii). Porém, com essa melhora, algumas perdas também aconteceram e são discutidas neste artigo.

APLICAÇÃO DE LOOP PIPELINING E LOOP UNROLLING À SÍNTESE DE ALTO NÍVELTÍTULO

Dione J. Ferrari[*], *Luiz Cláudio V. dos Santos*[+], *Rogério X. de Azambuja*[°] *e Felipe Vieira Klein*[#]

Departamento de Informática e Estatística
UFSC – universidade Federal de Santa Catarina
Florianópolis, Santa Catarina / Brasil

[*]dione@inf.ufsc.br [+]santos@inf.ufsc.br [°]xavier@inf.ufsc.br [#]fvklein@inf.ufsc.br

RESUMO

Grande parte do tempo de execução de aplicações é gasto com execução de laços, onde as instruções internas a este laço devem ser executadas inúmeras vezes. Dessa forma, a otimização do tempo de execução dessas estruturas, através de técnicas de paralelização, pode produzir resultados significativos.

Este artigo apresenta uma abordagem para a otimização dessas estruturas. Nossa abordagem baseia-se na aplicação de técnicas de paralelização de laços ("Loop Pipelining" e "Loop Unrolling") à Síntese de Alto Nível. São apresentados os resultados de experimentos utilizando modelos clássicos da Síntese de Alto Nível e também comparações destes com outros resultados reportados por outros autores da literatura.

Os resultados se mostraram promissores uma vez que as comparações mostraram que encontramos melhores intervalos de introdução de dados (dii). Porém, com essa melhora, algumas perdas também aconteceram e são discutidas neste artigo.

1. INTRODUÇÃO

A evolução dos sistemas computacionais está tornando cada vez mais comum o uso de *Sistemas computacionais Embutidos* em equipamentos que fazem parte do nosso cotidiano. Junto com essa evolução, há a competição de industrias no mercado de sistemas, que requer um projeto rápido para que as coloque na liderança desse mercado.

Por esses e outros motivos, Síntese de Alto Nível (HLS) têm sido uma área de pesquisas intensas para a produção de projetos VLSI. Tem sido mostrado que as ferramentas de HLS são uma boa escolha para os projetos de aplicações específicas como circuitos para processamento digital de sinais (DSP) e outros. Tradicionalmente, HLS tem sido utilizada para obter projetos rápidos e pequenos.

A HLS transforma uma especificação do comportamento de um sistema no nível algorítmico para uma arquitetura no nível RT (Regiter-Transfer) que implementa aquela especificação [1] e pode ser resumida em quatro etapas: *seleção de módulos*, que determina quantos módulos (unidades funcionais, elementos de memória e elementos de interconexão) de cada tipo serão necessários; *alocação*, responsável por quantos módulos de cada tipo serão necessários; o *escalonamento*, que define quando as operações são executadas; *ligação*, que define em qual módulo específico cada operação será executada.

Muitas vezes, grande parte do tempo de execução de uma aplicação é gasto em função da execução de laços, onde as instruções internas a este devem ser executas inúmeras vezes. Dessa forma, a otimização do tempo de execução dessas estruturas, através de técnicas de paralelização, pode produzir resultados significativos.

Apresenta-se neste artigo a utilização de técnicas que podem ser aplicadas no processo de HLS e que visam a otimização do tempo de execução de laços sob uma restrição de recursos (unidades funcionais).

Este artigo está organizado da seguinte forma: a Seção 2 apresenta a noção de paralelização de laços. Trabalhos anteriores com técnicas de paralelização de laços na HLS são apresentados na Seção 3.A Seção 4 apresenta a nossa abordagem para o problema de otimização através de exploração de soluções alternativas. A Seção 5 relata os resultados obtidos em nossos experimentos. Apresenta-se também uma comparação desses resultados com outros reportados na literatura. A Seção 6 conclui este artigo.

2. PARALELIZAÇÃO DE LAÇOS

Para se automatizar o processo de Síntese de Alto Nível é preciso que alguns modelos sejam definidos. O DFG ou grafo de fluxo de dados descreve o comportamento de um algoritmo na forma de um grafo, incorporando suas operações e suas dependências. A partir dele, dois grafos são obtidos. Um dos grafos, denominado DPG ("datapath graph"), modela a estrutura da unidade operativa (nível RT). O outro, denominado SMG ("state machine graph"), modela a unidade de controle. Em um SMG, cada vértice corresponde a um estado e as arestas representam as transições entre os estados.

Muitos sistemas possuem laços de instruções que devem ser executados inúmeras vezes e outros sistemas são totalmente executados em muitas iterações (filtro de ondas digitais). As técnicas de paralelização de laços partem do princípio de que instruções de diferentes iterações do laço possam ser executadas simultaneamente. Dessa forma, durante a execução da iteração i, uma instrução da iteração i+1 pode ser executada se houver recurso disponível para isso.

Antes de mostrarmos um exemplo de escalonamento de laços utilizando técnicas de paralelização de laços, algumas definições são necessárias.

Definição 1 – Um *grafo polar de fluxo de dados* DFG(V, E) é um grafo orientado onde cada vértice $v_i \in V$ representa uma operação e onde cada aresta $(v_i, v_j) \in E$ representa uma dependência de dados entre as operações v_i e v_j. Os pólos são vértices v_0 e v_n, denominados fonte e sumidouro.

Definição 2 - Um *grafo da unidade operativa* DPG(C, W) é um grafo não orientado onde cada vértice $c_i \in C$ representa um componente onde cada aresta $(w_i, w_j) \in W$ representa uma interconexão entre os componentes c_i e c_j.

Definição 3 - Um *vetor de restrições de recursos* a é um vetor onde cada componente a_k representa o número de recursos disponíveis de um determinado tipo $k \in \{1, 2, ..., R\}$.

Definição 4 – Uma *instância* de uma operação v_i na iteração m do laço, denotada por v_i^m, é uma réplica daquela operação que consome os valores dos operandos disponíveis na m-ésima iteração no laço.

Definição 5 – Sejam s_0, ..., s_i, ... , s_j, ... , s_n estados sucessivos do SMG. A transição (s_j, s_i), que define o corpo de

um laço cujo primeiro estado é s_i e cujo último estado é s_j, é denominada *back edge*.

Definição 6 – Sejam s_0, ..., s_i, ... , s_j, ... , s_n estados sucessivos do SMG tais que a aresta (s_j, s_i) seja do tipo "back edge". O *intervalo de introdução de dados* dii do laço é igual ao comprimento do caminho de s_i a s_j.

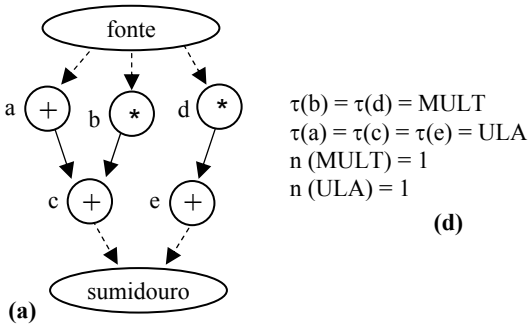
Definição 7 – Dado um SMG e um estado arbitrário $s_k \in S$, o conjunto das *instâncias disponíveis* em s_k, denotado por A_k, é o conjunto de todas as instâncias v_j^x que podem ser escalonadas no estado s_k.

Definição 8 – Dado um SMG e um estado arbitrário $s_k \in S$, o conjunto das *instâncias pendentes* em s_k, denotado por U_k, é o conjunto de todas as instâncias v_j^x que começaram a executar em um estado anterior e ainda precisam ser escalonadas.

Definição 9 – Dado um SMG e dois estados arbitrários s_i e $s_j \in S$, s_i é equivalente a s_j, escrito $s_i \equiv s_j$, se e somente $A_i = A_k$ e $U_i = U_k$

Definição 10 – Uma *janela* de tamanho W começando em uma dada iteração i é o conjunto de iterações compreendidas entre a iteração i a iteração i + W - 1.

A Figura 1a mostra um DFG simples que servirá de exemplo para o escalonamento utilizando esta técnica.



(a)

$\tau(b) = \tau(d) = MULT$
$\tau(a) = \tau(c) = \tau(e) = ULA$
n (MULT) = 1
n (ULA) = 1

(d)

$\prod = \{ b^i, d^i, b^{i+1}, d^{i+1}, a^i, e^i, e^{i+1}, c^i, c^{i+1}, a^{i+1} \}$

W = 2

(c)

Figura 1: DFG exemplo

Assuma que este DFG representa o corpo de um laço e que as operações são re-executadas em cada iteração deste laço. Assuma também que um somador e um multiplicador estão disponíveis e que as operações "a", "c" e "e" mapeiam para o somador enquanto as "b" e "d" mapeiam para o multiplicador, como indicado na Figura 1b. Para este exemplo, a execução de uma operação executada no recurso ALU leva um único ciclo e a execução de uma operação executada em MULT leva dois ciclos.

Por simplicidade, supõe-se que o laço execute um número infinito de vezes. Por isso, não se representa o teste de saída do laço para controlar a saída do laço.

No exemplo da Figura 1, assume-se uma regra de prioridade para selecionar instâncias que competem pelo mesmo recurso, a qual é codificada na forma de uma permutação de instâncias dentro de uma dada janela de iterações. A Figura 1c contém uma codificação de prioridade, que é uma permutação $\prod$ de instâncias das operações com tamanho de janela igual a dois (W=2). Na permutação $\prod$, i corresponde à menor iteração atualmente dentro da janela.

A Figura 2 mostra o processo de construção do SMG (escalonamento), estado por estado. para o exemplo da Figura 1. O conjunto de instâncias prontas para serem escalonadas e o conjunto de instâncias pendentes (levam mais de um estado para executar) para cada estado estão representados na Tabela 1. Em particular, dado o estado s_0, o conjunto A_0 contém instâncias de operações que dependem somente de valore de entrada, enquanto que o conjunto U_0 está vazio, pois nenhuma instância foi escalonada antes do estado atual (s_0). Esses conjuntos possuem instâncias ordenadas de acordo com a codificação de prioridade $\prod$.

Tabela 1: Os conjuntos A_k e U_k para o exemplo

Estado	Instâncias disponíveis	Instâncias pendentes
s_0	$A_0 = \{ b^0, d^0, b^1, d^1, a^0, a^1 \}$	$U_0 = \emptyset$
s_1	$A_1 = \{ d^0, b^1, d^1, a^1 \}$	$U_1 = \{ b^0 \}$
s_2	$A_2 = \{ d^0, b^1, d^1, c^0 \}$	$U_2 = \emptyset$
s_3	$A_3 = \{ b^1, d^1 \}$	$U_3 = \{ d^0 \}$
s_4	$A_4 = \{ b^1, d^1, e^0 \}$	$U_4 = \emptyset$
s_5	$A_5 = \{ d^1, b^2, d^2, a^2 \}$	$U_5 = \{ b^1 \}$

Note que no primeiro estado (s_0), de acordo com A_0, a instância a^0 e a instância b^0 são escalonadas e mapeadas para somador e multiplicador respectivamente. Um novo estado s_1 é criado.

No estado s_1, a instância b^0 está contida no conjunto U_1, ou seja, ela precisa ser escalonada neste estado (operação não finalizada) e por isso, é mapeada para o recurso multiplicador. Neste mesmo estado existe um somador livre e a^1 é escalonado neste estado (operações de diferentes iterações escalonadas no

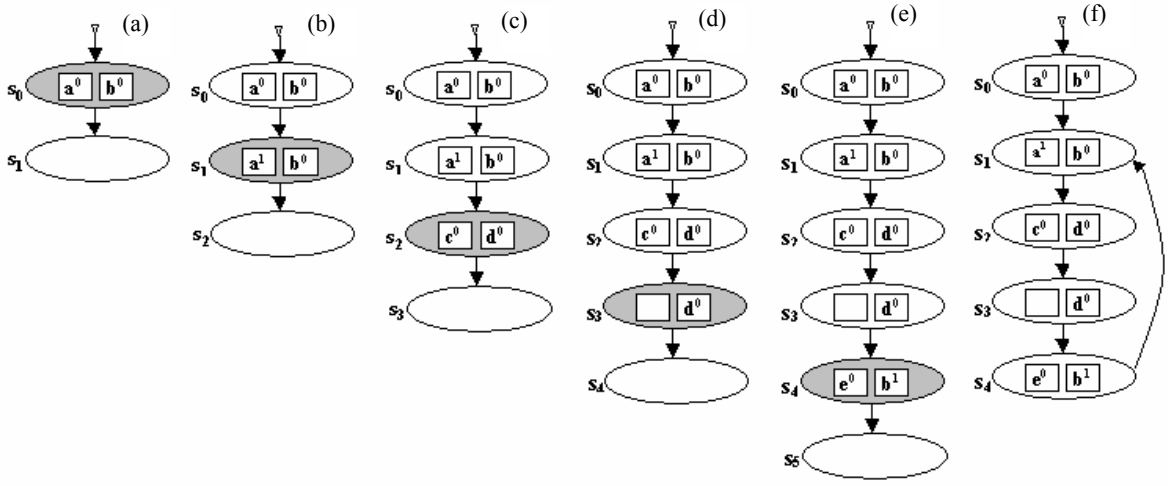


Figura 2: Escalonamento do laço da Figura 1.

mesmo estado).

A construção do SMG acontece dessa forma até que, no escalonamento do estado s_5, é detectada uma relação de estados equivalentes (veja Definição 9) entre os estados s_5 e s_1. Tal equivalência significa que o escalonamento dos estados s_5 e s_1 e de seus respectivo sucessores resultaria em idênticas seqüências periódicas de operações, que só se distinguem por pertencerem a diferentes iterações. Assim, ao invés de escalonar o estado s_5, insere-se uma transição do estado s_4 diretamente para o estado s_1, sendo o estado s_5 removido. Note que a transição (s_4, s_1) é uma aresta do tipo "back edge", o que caracteriza o fechamento do laço paralelizado. Dessa forma, o intervalo de introdução de dados encontrado é igual a 4 (distância entre os estados s_4 e s_1).

Note que o SMG final na Figura 2f contém um ciclo. Este ciclo possui instâncias de operações de diferentes iterações do laço, que está conseqüentemente paralelizado.

3. TRABALHOS ANTERIORES

Paralelização de Laços é uma idéia relativamente antiga. Atualmente, há vários algoritmos e modelos para paralelização. Dentre outros, podemos destacar *Modulo Scheduling* [2] e *Enhanced Pipeline Scheduling* [3].

Modulo Scheduling (MS) é uma das mais populares técnicas de paralelização de laços e, conseqüentemente, é utilizada como base para outros algoritmos.

Enhanced Pipeline Scheduling (EPS) modela o laço como uma seqüência cíclica de operações. O laço é paralelizado movendo operações no sentido contrário ao do ciclo, o que equivale a obter instâncias das operações movidas em iterações anteriores.

Outra alternativa é utilizar loop unrolling [4] como mecanismo básico. *Perfect Pipelining* (PP) trabalha da seguinte maneira: o laço é desenrolado um certo número de vezes e a partir daí, as operações são escalonadas até que um padrão seja detectado para o laço. Algumas regras são necessárias para garantir que se alcance um padrão. Uma desvantagem do PP é que em alguns casos, a convergência para um padrão pode ser muito demorada. Uma vantagem do PP é que a estrutura do algoritmo é tal que o procedimento de paralelização é separado do procedimento escalonador, o que evita que as heurísticas de escalonamento limitem o processo de paralelização.

Com a especificação de sistemas embutidos com restrições de tempo e de recursos cada vez mais severas, torna-se importante escolher um algoritmo de paralelização de laços adequado à exploração do espaço do projeto (*design space operation*). Isto motiva uma abordagem orientada à exploração de soluções alternativas. Tal abordagem é descrita a seguir.

4. NOSSA ABORDAGEM

A Seção 2 descreve um algoritmo, dentre os vários existentes na literatura, utilizados para resolver o problema de paralelização de laços. Em sua maioria, esses algoritmos possuem baixa complexidade, mas por utilizarem diferentes heurísticas, geralmente conflitantes, e gerarem uma única solução, não conseguem gerar soluções de boa qualidade sob restrições muito severas. Por outro lado, existem algoritmos que geram ótimas soluções, mas tornam-se inviáveis por apresentarem complexidade exponencial.

Para aumentar as chances de se encontrar uma boa solução sem utilizar algoritmos muito complexos, torna-se necessário não restringir o espaço de busca a uma única solução, ou seja, explorar diversas soluções na tentativa de otimizar os resultados, sem cair em uma busca exaustiva. Assim, existe um compromisso entre o tempo de busca e a qualidade da solução obtida.

É possível explorar soluções alternativas com a definição de uma codificação de prioridades para determinar em que ordem as operações disponíveis serão escalonadas [6]. A idéia-chave da abordagem de exploração de soluções é a seguinte: diferentes codificações de prioridades resultam em diferentes soluções com custos possivelmente distintos. Dessa forma, a otimização do custo é suportada pela monitoração dos custos de diferentes soluções exploradas (diferentes codificações) e a escolha da solução de menor custo. A codificação de prioridade usada neste trabalho baseia-se em permutações de instâncias das operações do DFG.

Nesse contexto, o algoritmo proposto por Aiken, Nicolau e Novack [5] parece suficientemente geral, garantindo a flexibilidade necessária para o tratamento de condicionais e laços, podendo ser facilmente adaptado para explorar soluções alternativas, conforma mostrado em

[6[6]. Por estas razões, tal algoritmo foi adotado neste trabalho.

A Figura 3 mostra a abordagem utilizada e pode ser resumida como a interação entre dois blocos principais: o explorador e o construtor. O *explorador* é o bloco encarregado da criação da codificação de prioridade $\prod$ e análise do custo da solução correspondente.O *construtor* gera uma solução para cada π e retorna o custo para o explorador.

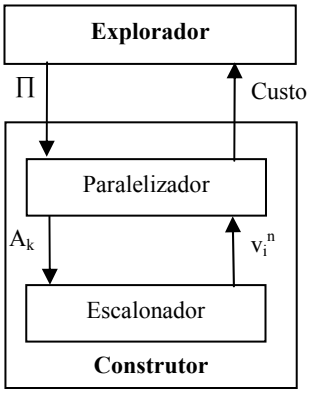


Figura 3: Visão geral da abordagem.

Pode-se observar a independência entre os blocos e a comunicação entre os mesmos através dos parâmetros π e custo. Isto torna a abordagem mais flexível, permitindo a adaptação do algoritmo em um dos blocos sem interferir na funcionalidade do outro e também facilita o controle do número de soluções que se deseja explorar.

O construtor consiste em um escalonador e um paralelizador. O paralelizador tem a função de capturar o paralelismo entre as operações da descrição comportamental e utiliza-lo na paralelização de laços. Para cada estado a ser escalonado, o paralelizador determina quais as instâncias estão disponíveis (Ak) e as envia para o escalonador. Este, baseado na codificação de prioridade, determina qual instância será escalonada e retorna esta para o paralelizador. Sabendo qual instância dentre as disponíveis deverá ser escalonada, o paralelizador marca esta instância como escalonada e atualiza o conjunto Ak. Isso se repete até não existirem mais instâncias passíveis de escalonamento. Então, o paralelizador cria um novo estado sk+1 e atualiza o seu respectivo conjunto Ak+1. Dessa forma, o SMG vai sendo construído pelo paralelizador passo a passo.

O escalonamento termina quando o paralelizador detectar uma relação de equivalência entre estados quando, é inserida uma aresta do tipo "back edge" entre o estado sk-1 e sj. Dessa forma, o paralelizador torna cíclico o SMG, o que representa a repetição de uma seqüência idêntica de instâncias

de operações pertencentes a iterações distintas, ou seja, representa um laço paralelizado.

5. RESULTADOS EXPERIMENTAIS

Nos experimentos foram utilizados dois exemplos clássicos, extraídos da literatura de Síntese de Alto Nível. O exemplo FDCT é um algoritmo que calcula a "fast discrete cosine transform" e foi retirado de [7]. O exemplo WDELF é um algoritmo que implementa um filtro de onda digital de quinta ordem e foi retirado de [8].

Os experimentos aqui descritos foram realizados sob a hipótese de que o atraso de execução de todas as operações é unitário, exceto as multiplicações, cujo atraso de execução é igual a dois ciclos de relógio.

As operações de adição e subtração podem ser executadas em um mesmo operador, denominado de unidade lógico-aritmética (abreviadamente ALU). Assim, diferentes tipos de operações podem ocupar o mesmo operador (em diferentes ciclos de relógio). Já as operações de multiplicação podem ocupar apenas o operador denominado multiplicador (abreviadamente MULT).

Foram pesquisadas em nossos experimentos 1000 soluções alternativas para os dois exemplos utilizados.

A Tabela 1compara nossos resultados com os reportados em [9] para o exemplo WDELF, sem paralelização. Para este exemplo, o custo ótimo foi obtido para todas as restrições de recurso testadas.

Tabela 1: dii para o exemplo WDELF (sem paralelização).

Restrição de Recursos		[9]		Nossa Abordagem	
MUL	ALU	Ótimo dii	LS dii	dii	λ
3	3	17	17	17	17
2	2	18	19	18	18
1	2	21	21	21	21
1	1	28	28	28	28

As duas primeiras colunas desta tabela representam as restrições de recursos. A terceira coluna mostra o menor dii que pode ser encontrado para uma dada restrição de recursos. A quarta coluna contém os resultados reportados na literatura [9] utilizando um algoritmo de "List Scheduling" (LS). As quinta e sexta colunas representam a melhor solução obtida em nossa abordagem ao se minimizar o dii.

Observe que trabalhando sem paralelização de laços, ou seja, W = 1, nossa abordagem obteve os valores ótimos para todas as restrições de recursos testadas, exceto para um dos casos. Isto demonstra que o algoritmo funciona correta e eficientemente, quando o paralelismo é restrito a uma única iteração.

A Tabela 2 apresenta uma comparação entre os resultados obtidos em nossa abordagem com os reportados na literatura usando paralelização. A duas primeiras colunas da tabela representam as restrições de recursos. A terceira e a quarta coluna mostram o dii e a latência, respectivamente, conforme relatados em [10]. De forma similar, a quinta e sexta colunas apresentam resultados reportados em [11]. As seis últimas colunas relatam o dii e a latência encontrados em nossa abordagem para diferentes tamanhos de janelas (W). É importante frisar que os resultados relatados em nossa abordagem correspondem às soluções com menor dii obtidas para cada restrição de recursos.

Observe que nossa abordagem obteve melhores valores de dii em todos os casos. Por exemplo, para a restrição de recursos de 2 multiplicadores e 3 unidades lógico-aritméticas,

Tabela 2: Comparação do dii para WDELF (com paralelização).

Restrição de Recursos		[10]		[11]		Nossa abordagem W=2		W=3		W=4	
MUL	ALU	dii	λ	dii	λ	dii	λ	dii	λ	dii	λ
3	4	19	16	-	-	11	21	7	24	7	31
3	3	-	-	16	18	10	22	9	30	9	41
2	3	17	17	16	18	11	24	9	32	9	43
2	2	18	18	17	19	13	29	13	42	13	57
1	2	20	20	19	21	16	37	16	55	16	72
1	1	29	28	-	-	26	49	26	78	26	109

obteve-se valores para dii de 11 (W=2) e 9 (W=3 e W=4), enquanto o melhor dii reportado na literatura foi de 16 [11]. Note, porém, que enquanto a latência relatada na literatura foi de 17, nossa abordagem obteve λ=24 (W=2), 32 (W=3) e 43 (W=4). Portanto, aparentemente, nossa abordagem obteve um menor dii às custas de uma maior latência.

A Tabela 3 relata experimentos de minimização de latência. Encontrou-se somente um autor que relatou em experimentos similares [9]. Note que, em todos os casos testados, encontraram-se valores de latência iguais ou melhores que os reportados em [9]. Todavia, em apenas um caso (A), encontrou-se um melhor dii.

Tabela 3: Comparação da latência para WDELF.

	Restrição de Recursos		[HEIJ96]		Nossa Abordagem (W = 2)	
	MUL	ALU	dii	λ	dii	λ
A	3	4	16	14	15	14
B	3	3	16	14	18	14
C	2	3	16	17	22	14
D	2	2	17	15	24	15
E	1	2	19	21	35	15
F	1	1	28	28	49	18

Neste trabalho, fez-se também uma análise do espaço de soluções. O Gráfico 1 representa o espaço de soluções para o WDELF, sob uma mesma restrição de recursos (6 MULTs e 6 ALUs). O espaço de soluções é representado em termos de dii e latência, para diferentes valores de W.

Note que para W = 2 tem-se um espaço de solução restrito e que este espaço aumenta à medida que aumentamos W. Isto ocorre pelo fato de expormos paralelismo entre mais instâncias. Note também que quando aumentamos W, o espaço de soluções alcança melhores valores de dii. Entretanto, o aumento de W tende a aumentar as latências. Isto reforça o fato de que melhores valores de dii foram encontrados às custas de uma pior latência. Em resumo, analisando o espaço de soluções para uma mesma restrição de recursos e diferentes W, pode-se concluir que os melhores valores de dii estão diretamente relacionados com o aumento da latência e com o aumento do espaço de soluções.

O Gráfico 2, também representa o espaço de soluções para o exemplo WDELF. Porém, este gráfico mostra os espaços de soluções obtidos para diferentes restrições de recursos, mas para uma janela de tamanho fixo (W=2).

Note que o espaço de soluções converge para valores cada vez melhores (tendendo para dii = 0 e latência = 0) à

Tabela 4: Casos de restrição de recursos

	Casos																	
	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**
MULs	1	1	2	2	3	4	4	3	4	5	5	6	7	6	8	10	12	14	16
ALUs	1	2	2	3	3	3	4	4	5	5	6	6	6	7	8	10	12	14	16

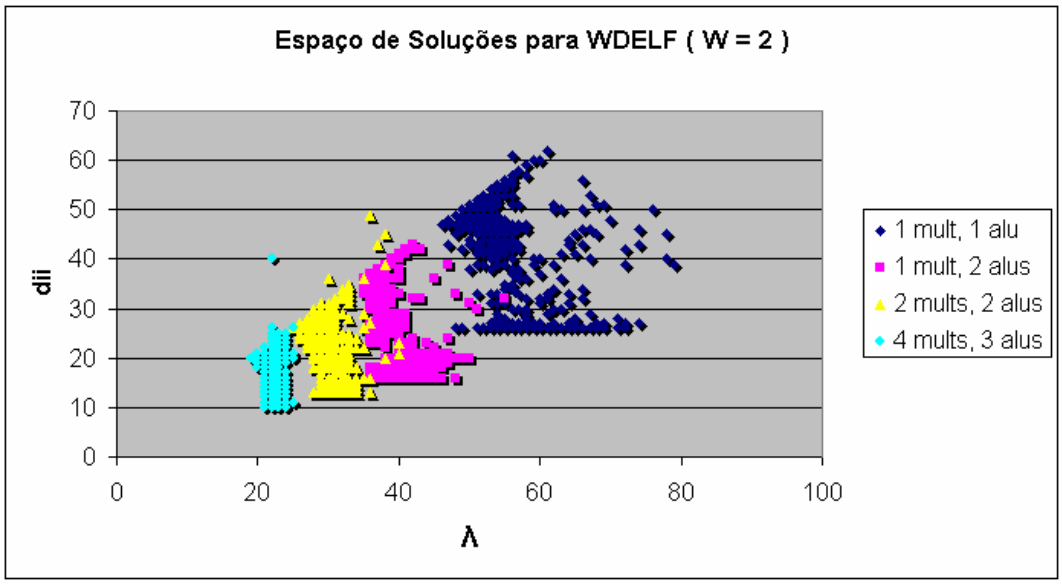


Gráfico 2: Espaço de soluções para WDELF (W = 2).

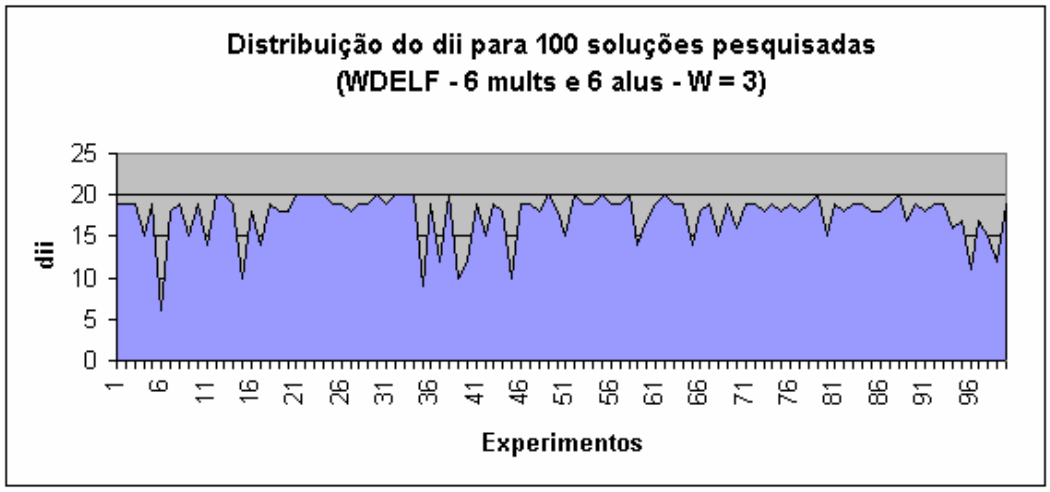


Gráfico 3: Distribuição do dii para 100 soluções pesquisadas (WDELF).

medida que aumentamos o número de recursos. Isso se deve ao fato de que à medida que aumentamos o número de recursos, as instâncias podem ser neles melhor acomodadas resultando em menor latência e dii.

O Gráfico 3 ilustra a distribuição do dii desde a primeira até a centésima solução pesquisada para o exemplo WDELF. Note que, apesar da geração de soluções alternativas permitir a exploração do espaço de projeto, evidencia-se aqui, que o processo de busca não mostra uma convergência nítida para soluções cada vez melhores. Isto é uma conseqüência da limitação atual da implementação do explorador.

O Gráfico 4 mostra o dii e a latência encontrados para o escalonamento de WDELF sob uma restrição de 6 MULTs e 6 ALUs para W = 2. A distribuição foi feita para diversas restrições de recursos (de A até S) conforme descrito na Tabela 4.

Note que para os casos de A à G, ao se aumentar os recursos, obteve-se uma melhora no dii e na latência, ou seja, melhores soluções foram encontradas com o aumento dos recursos. Entretanto, para os casos de H a S obteve-se piores resultados de dii ao se aumentar os recursos.Um comportamento similar pode ser observado no Gráfico 5, que é em tudo análogo ao anterior, mas apresenta resultados obtidos para W=3.

Observe que o ponto de inflexão da curva a partir do qual os valores de dii começam a piorar é distinto nos Gráficos 4 e 5. No Gráfico 4, tal ponto de inflexão corresponde à restrição G; no Gráfico 5, à restrição L. Isto se deve a diferentes balanceamentos entre o paralelismo exposto e o paralelismo acomodável nos recursos disponíveis. No Gráfico 4 o paralelismo exposto corresponde a instancias de duas iterações (W=2), o qual é totalmente acomodado nos recursos correspondentes à restrição G. Portanto, um aumento de recursos

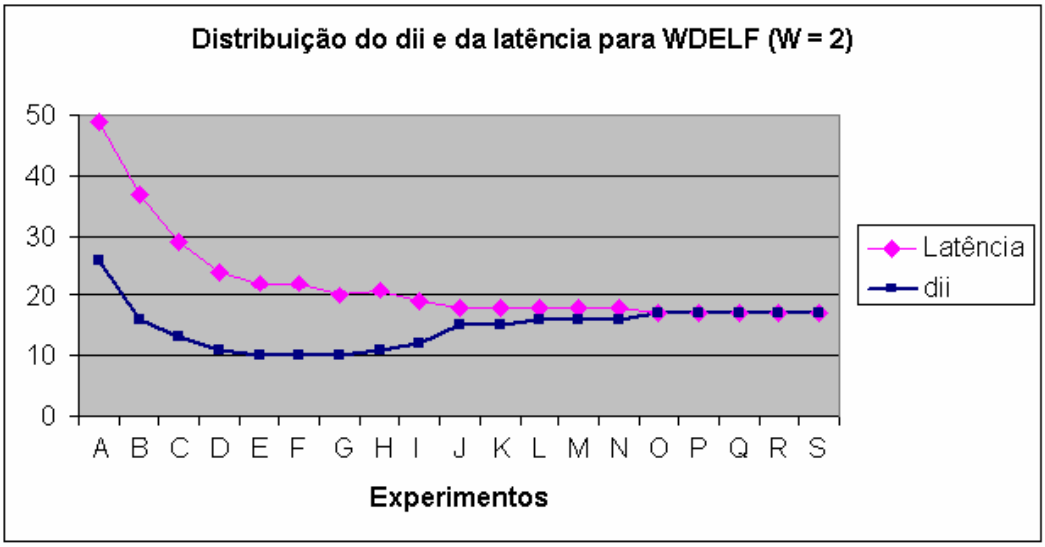


Gráfico 4: Distribuição do dii e da latência para WDELF (W = 2).

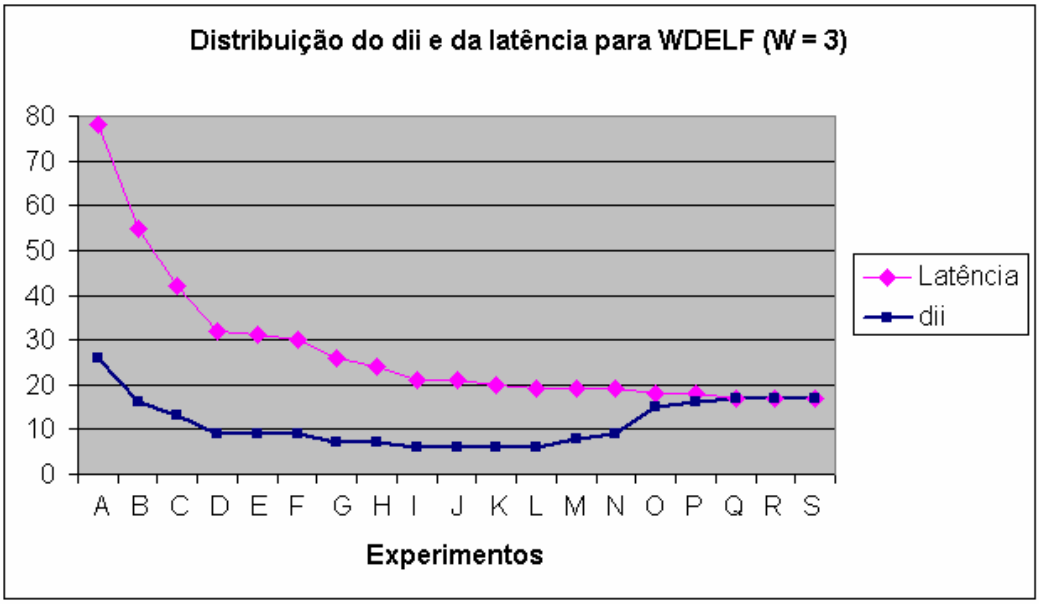


Gráfico 3: Distribuição do dii e da latência para WDELF (W = 3).

deveria produzir resultados melhores, uma vez que todo o paralelismo exposto já estaria acomodado. No Gráfico 5, o balanceamento do paralelismo ocorre para um maior número de recursos, correspondente à restrição L, porque mais recursos são necessários para acomodar o maior paralelismo exposto, que agora corresponde a instâncias de três iterações (W=3).

Todavia, o balanceamento por si só não explica o comportamento anômalo que leva à degradação do dii. Esse comportamento pode ser atribuído a uma deficiência do algoritmo de escalonamento utilizado no escalonador, ou seja, o "List Scheduling". A heurística do LS procura escalonar o maior número de instâncias possíveis em um estado, enquanto existirem recursos disponíveis. Isso faz com que o dii torne-se cada vez mais independente de Π, conforme os recursos vão se tornando mais abundantes. Essa independência ocorre porque quando existem recursos livres em número suficiente, todas as instâncias disponíveis são escalonadas, qualquer que seja a ordem delas em Π. Dessa forma, várias codificações de prioridade distintas resultam em escalonamentos idênticos. Felizmente, os projetos típicos têm restrições severas de recursos, fazendo com que essa deficiência tenda a não ser observada em casos práticos.

A Figura 1 mostra o corpo do laço paralelizado de um escalonamento do exemplo WDELF, gerado com paralelização de laços utilizando janela de 3 iterações (W=3), obtido sob uma restrição de recursos de 2 multiplicadores e 3 unidades lógico-aritméticas.

Nesta figura, as linhas horizontais separam os estados e, conseqüentemente, mostram em qual estado cada instância da operação é executada. Note que as instâncias que mapeiam para os multiplicadores estão em dois estados, já que elas necessitam de 2 ciclos para executarem. As linhas verticais pontilhadas delimitam as iterações no corpo do laço (representada na figura como iterações i, i-1, i-2, i-3 e i-4) enquanto as linhas horizontais delimitam os estados (representados na figura como os subíndices s_0, s_1, s_2, s_3 e s_4).

Note que, em cada estado, instâncias de diferentes iterações são escalonadas sempre respeitando o tamanho da

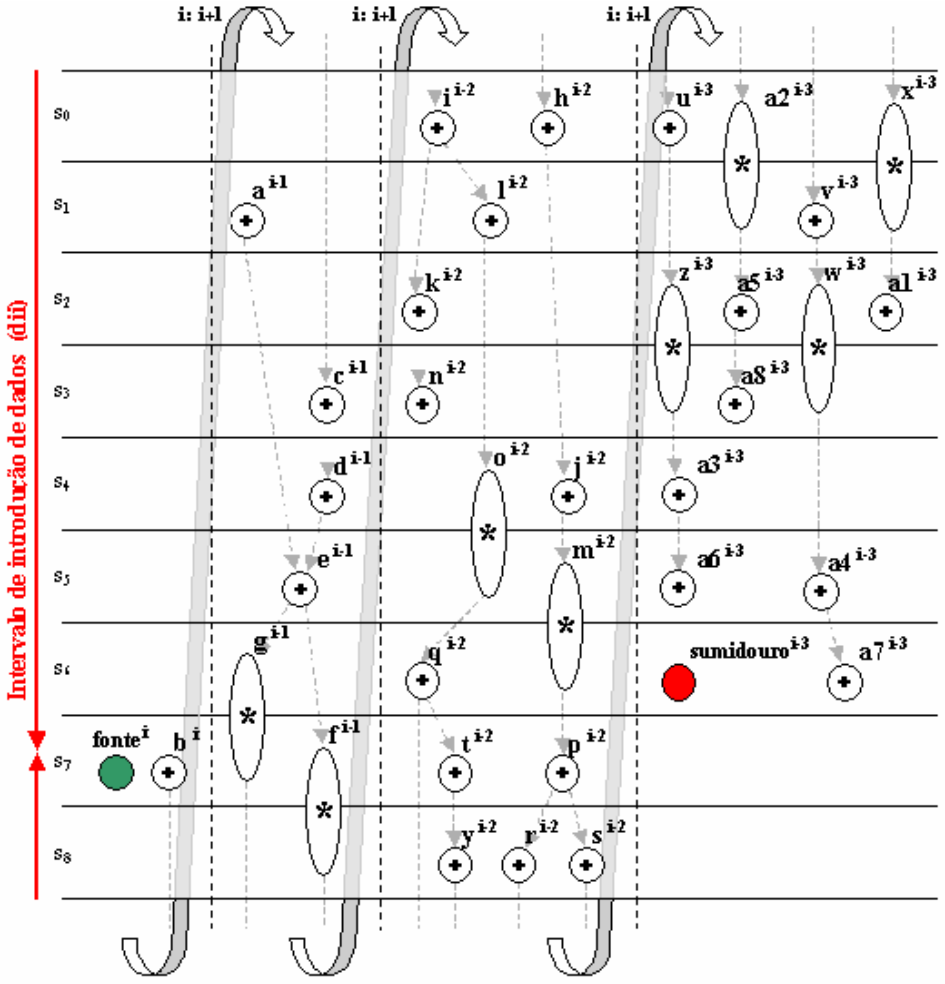


Figura 4: Resultado experimental ilustrativo de nossa abordagem ("pipelining").

janela. Por exemplo, no estado s_0 há instâncias das iterações i-1, i-2 e i-3 (W=3). Note também que o número de recursos não é excedido em estado algum.

6. CONCLUSÕES

Este artigo apresentou uma abordagem que permite a exploração de soluções alternativas para um problema clássico de Síntese de Alto Nível: o escalonamento com restrição de recursos. O problema-alvo foram as estruturas de repetição, que na maioria dos casos, tomam grande parte do tempo de execução de um programa. Para otimizar o escalonamento desses laços, técnicas da área de compiladores foram importadas, como por exemplo: "Loop Pipelining" e "Loop Unrolling".

Foram apresentados também resultados de experimentos e comparações destes com outros resultados reportados na literatura de Síntese de Alto Nível. Esses resultados se mostraram promissores uma vez que conseguimos encontrar melhores intervalos de introdução de dados para os laços. Porém, essa melhora no dii aconteceu às custas de um aumento da latência.

Fez-se também uma análise do espaço de solução em termos de dii e latência para avaliar a abordagem adotada neste trabalho.

8. REFERÊNCIAS BIBLIOGRÁFICAS

[1] Santos, L. C. V. dos, "A Síntese de Alto Nível na Automação de Projetos de Sistemas Computacionais", Capitulo 8 do livro-texto da VIII Escola de Informática da SBC-Sul, maio de 2000, pp. 211-231.

[2] Lam, M., "Software Pipelining: an effective scheduling technique for VLIW machines", Proc. SIGPLAN'88 Conf. on Programming Languages Design and Implementation, junho de 1998, pp. 318-328.

[3] Ebcioglu, K., "A Compilation Technique for Software Pipelining of Loops with Conditional Jumps", Proc. 20th Annual Workshop on Microprogramming, dezembro de 1987, pp. 69-79.

[4] Hennessy, J. L., Patterson, D. A., "Computer Architeture – A quantitative approach", 2nd edition, Morgan Keufmann Publisher inc., 1996.

[5] Aiken, A., et al, "Perfect Pipelining: A New Loop parallelization technique", Proc. European Symposium on Programming, pp. 221-235, 1998.

[6] Santos, Luiz C. V. dos, "Exploiting Instruction-level Parallelism: A Constructive Approach", Eindhoven University of Technology, PhD. Thesys, 1998.

[7] Mallon, D. J., Denyer, P. B., “A New Approach to Pipeline Optimization”, Proc. EDAC’90m oo, 1990.

[8] Dewilde et al., “Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms”, in S.Y. Kung, H. J. Whitehouse and T. Kailath, VLSI and Modern Signal Processing, Prentice Hall, 1985.

[9] Heijligers, M. J. M., “The Application of Genetic Algorithms to High-Level Synthesis”, PhD. Thesis, Eindhoven University to Technology, The Netherlands, 1996.

[10] Fransen, F., “Retiming for Dataflow Graphs, Training Report”, Eindhoven University of Technology, October 1994.

[11] Radivojevic, I., Brewer, F., “A New Symbolic Technique for Control Dependent Scheduling”, IEEE Transactions on Computer-Aided Design, vol. 15, nº 1, 1996.