POLITECNICO DI MILANO
Facoltï¿½ di Ingegneria dell'Informazione
Corso di laurea in Ingegneria Informatica

La Sintesi ad Alto Livello: Estensioni al Force Directed Scheduling

Relatore: Prof. Fabrizio FERRANDI

Tesi di laurea di
Marco LATTUADA
Matr. 666801

Anno Accademico 2005/2006

Alla mia famiglia

Indice

1 Introduzione **1**

2 Analisi dello Stato dell'Arte **5**

2.1 La Sintesi ad Alto Livello 6

2.1.1 Fasi della Sintesi ad Alto Livello 6

2.1.2 Livelli di Descrizione-Astrazione 8

2.1.2.1 Livello circuito 11

2.1.2.2 Livello logico 11

2.1.2.3 Livello microarchitetturale o RT 11

2.1.2.4 Livello algoritmico-funzionale 11

2.1.2.5 Livello sistema 12

2.1.3 Linguaggi di specifica del sistema 12

2.1.3.1 VHDL 13

2.1.3.2 SystemC 14

2.1.4 Rappresentazioni intermedie 16

2.1.4.1 Data Flow Graph 17

2.1.4.2 Control Flow Graph 17

2.1.4.3 Data Dependence Graph 18

2.1.4.4 Control Dependence Graph 19

2.1.4.5 System Dependence Graph 19

2.1.4.6 Control Data Flow Graph 20

2.1.4.7 Grafo dei Blocchi Basici 21

2.1.4.8 Albero dei Dominatori 22

2.2 Il problema dello scheduling 22

2.2.1 Algortimi che minimizzano metriche temporali 24
2.2.1.1 ASAP 25
2.2.1.2 ALAP 25
2.2.1.3 Path–Based 26
2.2.1.4 List Based 27
2.2.1.5 Static List 27
2.2.2 Algortimi che minimizzano metriche architetturali 28
2.2.2.1 Programmazione Lineare Intera 29
2.2.2.2 Simulated Annealing 31
2.2.2.3 Force Directed 31
2.2.2.4 Rescheduling Iterativo 32
2.3 La tecnica del Backtracking 33
2.3.1 Tecniche di Look-Ahead 34

3 Il Force Directed Scheduling proposto da Paulin e Knight 35
3.1 L'algoritmo nella sua versione base 36
3.1.1 Calcolo delle somme di probabilità 38
3.1.2 Calcolo delle Forze 41
3.1.3 Corpo dell'algoritmo 45
3.1.4 Considerazioni sulla complessità 45
3.2 Estensioni all'algoritmo proposte da Paulin e Knight 47
3.2.1 Scheduling con cicli 47
3.2.2 Estensioni allo scopo dell'algoritmo 48
3.2.2.1 Minimizzazione dei costi relativi ai bus 48
3.2.2.2 Minimizzazione dei costi relativi ai registri 49
3.2.3 Integrazione di informazioni relative all'architettura . . . 51
3.2.4 Rilassamento di alcune condizioni per l'applicazione dell'algoritmo 53
3.2.4.1 Scheduling con Chaining 53
3.2.4.2 Scheduling di operazioni multiciclo 54
3.2.4.3 Scheduling su unità pipelined 55
3.2.5 Estensione con vincoli sulle risorse 56
3.2.5.1 Il *force-directed list scheduling* 56

3.2.5.2 Calcolo del Numero di Passi di Controllo per Tentativi 58
3.2.6 Introduzione di vincoli temporali locali 59
3.2.7 Tecniche di look-ahead 59

4 Il Force Directed Scheduling proposto 63
4.1 Proposte di modifiche all'algoritmo originale 63
4.1.1 Priorità delle operazioni 64
4.1.2 Interpretazione sulla formula della forza 66
4.1.3 Correzione nel calcolo di *predecessors' and successors' forces* 73
4.1.4 Scelta del prossimo assegnamento da effettuare 74
4.1.5 Gestione dei cicli 78
4.1.6 Complessità dell'Algoritmo modificato 80
4.2 Estensione all'algoritmo con introduzione dei vincoli sulle risorse 80
4.2.1 Introduzione dei vincoli 81
4.2.2 Introduzione del binding su un tipo di unità funzionale . 86
4.2.3 Introduzione del Backtracking 92
4.2.3.1 Taglio di sottoalberi esplicitamente "morti" o "moribondi" 93
4.2.3.2 Taglio di sottoalberi implicitamente morti 94
4.2.3.3 Assegnamenti forzati 96
4.2.3.4 Backtracking Anticipato 97
4.2.3.5 Grado dell'albero di Ricerca 98
4.2.4 La scelta del prossimo assegnamento da effettuare 99
4.2.5 Riflessioni sulla priorità delle operazioni per questa versione dell'algoritmo 100
4.2.6 Analisi della complessità dell'Algoritmo 102

5 L'Implementazione all'interno del Progetto PandA 105
5.1 PandA Project 105
5.1.1 Analizzatore delle specifiche 106
5.1.2 Costruttore dei grafi 106
5.1.3 Lettore delle informazioni sulla tecnologia e sui vincoli . . 107

5.1.4 Schedulatori . . . 107
5.1.5 Sintetizzatore dell'unità di controllo . . . 108
5.2 Implementazione del Force Directed Scheduling . . . 108
5.2.1 Scelte implementative . . . 108
5.2.2 Diagrammi di classe e Diagrammi di Collaborazione . . . 109
5.2.2.1 stackSchedule . . . 110
5.2.2.2 FDschedule . . . 111
5.2.2.3 infOp_type . . . 112
5.2.2.4 job . . . 113
5.2.2.5 force_directed . . . 114
5.2.3 Calcolo delle somme di probabilità . . . 118
5.2.4 Calcolo della mutua esclusione fra le operazioni . . . 119

6 Risultati Sperimentali 121
6.1 Dimensioni dei benchmark . . . 123
6.2 Variazioni nel calcolo della forza . . . 123
6.3 Introduzione della priorità delle operazioni . . . 125
6.4 Cambio del criterio di scelta del prossimo assegnamento . . . 125
6.4.1 Test senza vincoli sulle risorse . . . 126
6.4.2 Test con vincoli sulle risorse . . . 126
6.5 Scheduling con vincoli sulle risorse . . . 127
6.6 Confronto con i risultati del List Based in caso di vincoli sulle risorse . . . 128
6.6.1 Diminuzione delle unità funzionali soggette a vincoli . . . 129
6.6.1.1 Benchmark Kim . . . 129
6.6.1.2 Benchmark Maha . . . 131
6.6.2 Diminuzione delle unità funzionali non soggette a vincoli 131
6.6.2.1 Benchmark Bandpass . . . 131
6.6.2.2 Benchmark Diffeq . . . 132
6.6.2.3 Benchmark Ewf_v2 . . . 132

7 Conclusioni e possibili sviluppi futuri 133

Riferimenti bibliografici 139

Elenco delle tabelle

6.1 Numero di operazioni e di salti condizionati di ogni benchmark . 123
6.2 Tabella comparativa calcolo Forza originale - calcolo Forza modificato nei test senza costrutti condizionali 124
6.3 Tabella comparativa calcolo Forza originale - calcolo Forza modificato nei test con costrutti condizionali 124
6.4 Tabella comparativa Force Directed senza priorità - Force Directed con Priorità 125
6.5 Tabella comparativa criterio di scelta del prossimo assegnamento - casi senza vincoli 126
6.6 Tabella comparativa criterio di scelta del prossimo assegnamento - casi con vincoli 127
6.7 Tabella comparativa nel caso di scheduling con vincoli sulle risorse 128
6.8 Unità allocate per il benchmark Kim 129
6.9 Unità allocate per il benchmark Maha 131
6.10 Unità allocate per il benchmark Bandpass 131
6.11 Unità allocate per il benchmark Diffeq 132
6.12 Unità allocate per il benchmark Ewf_v2 132

Elenco delle figure

2.1 Livelli di astrazione e domini di rappresentazione 9

3.1 Esempio di interpretazione dell'algoritmo del Force Directed scheduling 37
3.2 Esempio delle limitazioni insite nel calcolo delle *successors' and predecessors' forces* 44

4.1 Esempio di indecisione nella scelta di un assegnamento in caso di costrutti condizionali 67
4.2 Esempio di scelta di assegnamento svantaggiosa in caso di costrutti condizionali 68
4.3 Esempio di Operazione con Forze solo Positive 77

5.1 Diagramma della classe stackSchedule 111
5.2 Diagramma della classe FDschedule 112
5.3 Diagramma della classe infOp_type 113
5.4 Diagramma della classe job 114
5.5 Diagramma della classe force_directed 117

Capitolo 1

Introduzione

> *"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year ... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer"*
>
> Gordon E. Moore, 1965 [Moo65]

Negli ultimi decenni si è verificato quanto anticipato da Moore a proposito dello sviluppo delle tecniche di integrazione di componenti digitali su silicio. Questo ha comportato un aumento considerevole della complessità del progetto di sistemi dedicati: a causa di ciò, a differenza degli albori della progettazione digitale, non è più possibile delegare completamente al progettista o ad un gruppo di essi l'intera attività di progettazione. Per questo motivo sempre più importanza rivestono gli strumenti di sviluppo automatico per la progettazione, strumenti che partendo da una specifica formale delle funzionalità del sistema generino la descrizione di una sua implementazione sintetizzabile.

Un altro aspetto critico nella progettazione di sistemi digitali che è emerso soprattutto negli ultimi anni è quello dell'ottimizzazione d'area e di potenza, a causa della grande diffusione che stanno avendo i sistemi portatili. Tuttavia il problema di minimizzare le metriche relative a questi aspetti è in contrasto con un altro aspetto significativo delle caratteristiche di un implementazione che

consiste nelle prestazioni: solitamente si fissano dei vincoli relativi alle metriche di un aspetto e si cerca di minimizzare quelle relative all'altro aspetto.

Questo lavoro si inserisce proprio all'interno di uno strumento di sviluppo automatico ed in particolare nella sua parte dedicata alla sintesi ad alto livello. All'interno di essa si è posta l'attenzione sulle fasi di scheduling e di allocazione delle risorse: questi due aspetti ed in special modo il secondo influenzano fortemente la qualità dell'implementazione relativamente alle metriche riguardanti area e potenza dissipata. Si è scelto quindi di lavorare sul Force Directed scheduling, in quanto esso pur essendo un algoritmo di scheduling incorpora al suo interno la funzione di allocazione delle risorse.

Il contributo portato da questo lavoro di tesi consiste proprio in una serie di modifiche applicate alla versione originale del Force Directed scheduling ed in particolare:

- modifiche atte a migliorare la qualità dei risultati prodotti (numero di unità funzionali allocate);
- modifiche atte a migliorare il tempo di computazione dell'algoritmo tramite diminuzione della sua complessità;
- introduzione della gestione di vincoli sul numero massimo di unità allocabili per i diversi tipi di risorse;
- introduzione della scelta del tipo di unità funzionale a cui assegnare un'operazione nel caso esistano più tipi che la possono eseguire.

Con queste modifiche, in particolare con le ultime due, si permette all'algoritmo di considerare contemporaneamente vincoli relativi a due tipi diversi di metriche, quali quelle relative al numero di unità funzionali allocate e quelle relative alle prestazioni e minimizzare quelle del primo tipo. La differenza rispetto a molti degli approcci classici consiste proprio nel considerare la possibilità di vincolare anche parametri relativi alla metrica che si vuole minimizzare.

La tesi è suddivisa in sette capitoli; il secondo capitolo propone l'analisi dello stato dell'arte relativo alla sintesi ad alto livello. In particolare vengono analizzate le diverse fasi che costituiscono questo processo soffermandosi sui

linguaggi utilizzati per la descrizione della specifica e sulle rappresentazioni intermedie utilizzate nel processo di sintesi. Vengono poi illustrati i diversi algoritmi di scheduling proposti in letteratura e presentata brevemente la tecnica del Backtracking.

Il terzo capitolo descrive in modo critico la versione originale dell'algoritmo del Force Directed scheduling proposto da Paulin e Knight compreso di estensioni.

Il quarto capitolo mostra le modifiche proposte all'algoritmo stesso che si riferiscono sia alla sua versione base sia all'introduzione di vincoli sulle risorse nel problema dello scheduling da risolvere.

Nel quinto capitolo viene fornita una breve descrizione di PandA, il framework all'interno del quale è stato implementato l'algoritmo proposto, e dell'implementazione realizzata sottolineando quali parametri sono stati scelti.

Nel sesto capitolo sono riportati i risultati sperimentali relativi alle diverse versioni dell'algoritmo, a partire da quella originaria, applicate su diversi benchmark per evidenziare i guadagni in termini di qualità di risultati o di tempi di computazione forniti dalle diverse modifiche proposte.

L'ultimo capitolo riporta l'analisi conclusiva del lavoro svolto evidenziando i possibili sviluppi futuri.

Capitolo 2

Analisi dello Stato dell'Arte

In questo Capitolo verrà mostrata una panoramica sullo stato dell'arte relativo alla Sintesi ad Alto Livello, con particolare attenzione agli algoritmi di scheduling.

Nella prima parte verrà data una breve descrizione di che cosa si intenda per Sintesi ad Alto Livello, illustrando le fasi principali in cui questo processo può essere suddiviso e soffermandosi su alcuni aspetti particolari quali i linguaggi per formalizzare la descrizione comportamentale di un sistema che ne costituisce il dato in ingresso, le rappresentazioni intermedie da esso utilizzate e come viene modellizzato il prodotto di tale processo.

Nella seconda parte verranno elencati i principali algoritmi di scheduling presenti in letteratura, raggruppati in base alle loro caratteristiche, e di ciascuno di essi verranno fornite le caratteristiche principali. Una descrizione molto più accurata dell'algoritmo Force Directed scheduling e una sua analisi verranno forniti nel capitolo 3 in quanto esso costituisce la base di partenza dell'algoritmo di scheduling sviluppato in questo lavoro.

Nell'ultima parte verrà data una breve descrizione della tecnica del Backtracking che è anch'essa utilizzata all'interno del lavoro presentato.

2.1 La Sintesi ad Alto Livello

La Sintesi ad alto livello è un processo che partendo da una descrizione comportamentale di un sistema, eventualmente corredata da vincoli, costruisce una descrizione dello stesso sistema a livello RT, quindi facilmente implementabile (per i diversi livelli di descrizione di un sistema confrontare 2.1.2). L'obiettivo che si vuole raggiungere utilizzando questo tipo di processo all'interno della progettazione di sistemi dedicati consiste nell'alzare il livello di astrazione a cui deve lavorare il progettista stesso tramite la realizzazione di strumenti automatici che gestiscano o semplifichino il problema della progettazione, dell'intero sistema o di alcune sue parti, ad un livello via via sempre più elevato. Maggiore sarà il livello a cui potrà lavorare lo strumento automatico, minore sarà l'attività lasciata al progettista.

Oltre a permettere un più rapido sviluppo del progetto, un secondo aspetto positivo della Sintesi ad Alto Livello, se correttamente eseguita, è il consentire l'automatizzazione parziale o totale della verifica dell'implementazione del sistema, diminuendo così la probabilità di errore e riducendo non solo il tempo necessario alla fase stessa di verifica, ma anche quello relativo alla fase di debugging.

2.1.1 Fasi della Sintesi ad Alto Livello

E' possibile suddividere la Sintesi ad Alto Livello in una serie di fasi principali che verranno eseguite in sequenza. Il flusso di progetto tuttavia non è obbligatoriamente lineare, ma è possibile che alcune singole fasi, o addirittura alcune successioni di esse vengano ripetute per compiere raffinamenti successivi all'implementazione nel caso quelle particolari fasi o quelle immediatamente successive non riescano a fornire risultati soddisfacenti:

1. Compilazione

   La descrizione comportamentale del sistema viene tradotta in una serie di rappresentazioni intermedie per essere meglio gestita dalle fasi successive.

2. Ottimizzazione

   Vengono individuate ed eseguite trasformazioni sulla descrizione intermedia del sistema atte a semplificarlo o a renderlo più facilmente trattabile (sono le ottimizzazione tipiche svolte dai compilatori).

3. Analisi del problema e costruzione dell'architettura

   In questa fase dalle rappresentazioni intermedie vengono estratte o calcolate le informazioni necessarie alla sintesi del datapath e del controllore. Essa si articola in tre sottofasi che a seconda degli algoritmi utilizzati possono essere eseguite contemporaneamente o in sequenza:

   a Allocazione

   Vengono definiti numero e tipo delle risorse: funzionali, di memorizzazione, e di comunicazione; per fare ciò è necessario trovare un equilibrio fra l'ottimizzazione dei costi che spinge ad utilizzare poche risorse relativamente economiche e l'ottimizzazione delle prestazioni che spinge ad utilizzare un numero maggiore di risorse più performanti e quindi solitamente più costose. Il costo non è necessariamente il costo economico dei singoli componenti, ma può essere per esempio l'area occupata dall'implementazione o la potenza consumata durante il suo uso. Per poter affrontare correttamente il problema è necessario avere stime più o meno accurate delle metriche in gioco relative ai diversi componenti.

   Esistono due principali approcci:

   - *Approcci costruttivi*: le unità funzionali, di memorizzazione e di interconnessione vengono allocate mentre viene visitato il CDFG (la sua descrizione è in 2.1.4.6) seguendo l'ordine delle dipendenze. Esempi di questo approccio sono forniti da [HT83] e [KP90]. Le soluzioni ottenute con queste tecniche possono discostarsi molto da quelle ottime;
   - *Approcci decompositivi*: il problema dell'allocazione viene scomposto in parti diverse, viene risolto per ciascuna di esse ed infine viene

prodotta la soluzione complessiva. Esempi ne sono forniti in [PK89] e [CD86].

b Scheduling

Il tempo di esecuzione della funzionalità viene suddiviso in intervalli uguali chiamati passi di controllo o cicli di clock; le singoli operazioni vengono assegnate ai diversi passi di controllo. I possibili approcci a questo problema verranno mostrati in 2.2.

c Binding

Consiste nel mappare elementi presenti nella descrizione del sistema su componenti architetturali dopo aver scelto il tipo delle interconnessioni (multiplexer piuttosto che bus). In particolare consiste nel mapping di:

- Variabili su Unità di Memorizzazione
- Operazioni su Unità Funzionali
- Trasferimenti di dati o comunicazioni fra unità funzionali o di memorizzazione su interconnessioni.

4. Sintesi del controllore

In quest'ultima fase viene sintetizzato il controllore modellizzato solitamente come macchina a stati finiti.

2.1.2 Livelli di Descrizione-Astrazione

Il sistema ed in particolare il circuito che lo implementa possono essere descritti in modalità diverse. Ogni modalità è caratterizzata da due aspetti: il dominio di rappresentazione utilizzato (ovvero il particolare punto di vista che si adotta) e il livello di astrazione (ovvero il dettaglio utilizzato nella descrizione stessa). Le possibili modalità di rappresentazione sono esemplificate dalla figura 2.1: gli assi rappresentano i diversi domini di rappresentazione mentre le circonferenze rappresentano i diversi livelli di astrazione, con l'accuratezza che cresce avvicinandosi al centro dei cerchi.

I domini di rappresentazione del circuito ([ea92]) sono essenzialmente tre:

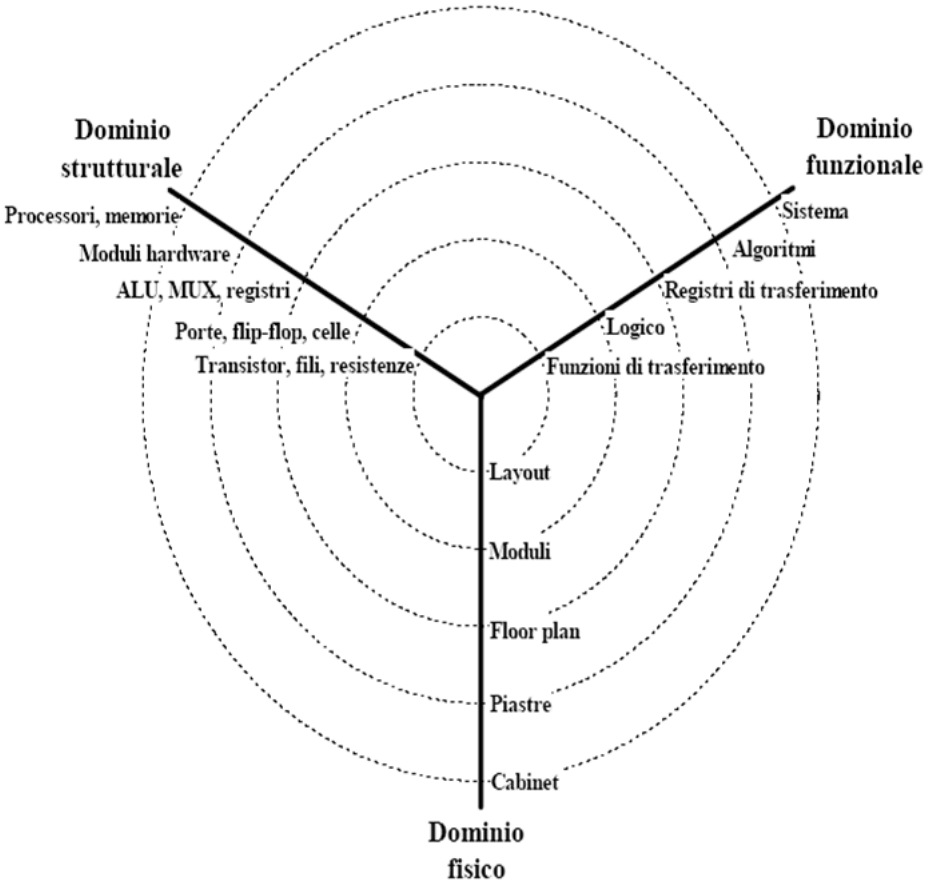


Figura 2.1. Livelli di astrazione e domini di rappresentazione

- Funzionale

  In questo dominio la descrizione fornisce solo le funzionalità fornite dal circuito e non la sua struttura; il circuito, o una sua frazione viene rappresentato come una scatola nera di cui vengono forniti solo ingressi, uscite e come queste dipendano dagli ingressi ovvero la funzione che esso implementa. Oltre alla funzionalità implementata riveste grande importanza in questo tipo di rappresentazioni la descrizione della sua interfaccia cioè le relazioni temporali fra i diversi segnali di ingresso e uscita.

- Fisico

  In questo dominio la descrizione fornisce solo la struttura fisica del circuito ignorando le funzionalità implementate. Vengono specificate la posizione nello spazio dei diversi componenti e la struttura delle interconnessioni.

- Strutturale

  Le rappresentazioni appartenenti a questo dominio costituiscono un compromesso fra quelle appartenenti agli altri due domini. In genere sono utilizzate nel passaggio fra il primo e il secondo tipo di rappresentazione. Infatti è possibile per il passaggio Funzionale-Strutturale far corrispondere a molti dei componenti della prima rappresentazione dei componenti della seconda e tale corrispondenza è univoca sotto predeterminati vincoli tecnologici. Per passare dalla rappresentazione Strutturale a quella Fisica invece sono in genere necessari due passi: nel primo passo vengono disposti in maniera approssimativa i diversi elementi circuitali curando il posizionamento delle interconnessioni, nel secondo ai componenti circuitali si sostituisce la realizzazione fisica di essi.

I livelli di astrazione delle descrizioni, ordinati in modo crescente (cui corrisponde un ordine decrescente di dettaglio) sono invece cinque. Per ciascuno di essi verrà brevemente presentata la descrizione relativa ad ogni dominio di rappresentazione:

2.1.2.1 Livello circuito

dominio funzionale: la descrizione è realizzata attraverso funzioni di traferimento e diagrammi temporali;

dominio strutturale: gli elementi base utilizzati nella descrizione sono transistor, resistenze, fili di interconnessione;

dominio fisico: la rappresentazione utilizzata per la descrizione è quella tipica di un layout di un circuito integrato.

2.1.2.2 Livello logico

dominio funzionale: si utilizzano espressioni booleane;

dominio strutturale: gli elementi base sono porte logiche, flip-flop e celle;

dominio fisico: vengono descritti posizionamento di celle e moduli nello spazio.

2.1.2.3 Livello microarchitetturale o RT

dominio funzionale: la descrizione utilizza registri e macchine a stati finiti;

dominio strutturale: i componenti della descrizione sono le diverse unità funzionali: unità aritmetico-logiche, multiplexer, registri, microcontrollori e micromemorie;

dominio fisico: viene descritto il posizionamento dei componenti elencati nello spazio.

2.1.2.4 Livello algoritmico-funzionale

dominio funzionale: la funzionalità è descritta attraverso un algoritmo in cui sono definite strutture dati ed operazioni che le devono manipolare; non è necessario che vi sia poi corrispondenza fra struttura dati e componenti architetturali così come fra operazioni e unità funzionali o unità di

controllo; possono essere utilizzati per la descrizione anche linguaggi di descrizione dello hardware;

dominio strutturale: i componenti di base sono gli stessi del livello microarchitetturale, ma raggruppati in macroblocchi quali linee di trasferimento dati, unità di controllo e unità di memorizzazione per sottolineare gli elementi di comunicazione e sincronizzazione fra i diversi componenti;

dominio fisico: gli elementi utilizzati nella descrizione sono le schede.

2.1.2.5 Livello sistema

dominio funzionale: viene descritta la funzionalità specificandone i vincoli, in particolare quelli relativi alle prestazioni; si tralascia qualsiasi aspetto implementativo;

dominio strutturale: gli elementi utilizzati nella descrizione sono processori, memorie, unità di controllo, switch e bus;

dominio fisico: viene descritto il posizionamento e la suddivisione del circuito a livello di gruppi di schede o di armadi.

2.1.3 Linguaggi di specifica del sistema

I linguaggi di specifica sono linguaggi utilizzati per fornire una descrizione comportamentale del sistema che abbia caratteristiche di formalità. Tali linguaggi inoltre consentendo un diverso livello di astrazione possono essere utilizzati anche per le rappresentazioni intermedie durante il processo di Sintesi ad Alto Livello. I linguaggi più utilizzati sono i linguaggi di descrizione dello hardware (HDL) e SystemC che può essere considerato una loro evoluzione.

I linguaggi della descrizione dello hardware sono stati introdotti per favorire lo sviluppo di sistemi hardware permettendo di descrivere con uno stesso linguaggio le caratteristiche del sistema a diversi livelli d'astrazione: funzionale, transazionale, architetturale o logico. E' quindi possibile utilizzare un unico linguaggio come rappresentazione intermedia nelle diverse fasi della Sintesi

ad Alto Livello. Inoltre è possibile far convivere nella stessa rappresentazione componenti descritti a livelli di astrazione e quindi di dettaglio diversi. Per questo motivo è possibile raffinare la descrizione solo di alcune parti del sistema mantenendo a grana grossa quella degli altri componenti.

Il primo HDL è stato l'*ISP* formalizzato da Gordon Bell e Alan Newell e descritto in [Bel71]. In questo linguaggio veniva per la prima volta introdotta la descrizione a livello RT (register transfert level). Infatti l'ISP era utilizzato per la descrizione del comportamento del computer PDP-8 che veniva modellizzato come un insieme di registri e un insieme di funzioni logiche che descrivevano il trasferimento dati tra di essi.

In seguito nacquero diversi altri linguaggi di descrizione dello hardware: *Vhsic HDL (VHDL)*, *UDLI* sviluppato da NTT, *HiLo*, da cui è stato derivato *Verilog* e *ISP'*, derivato di ISP.

2.1.3.1 VHDL

VHDL è stato sviluppato dal governo degli Stati Uniti d'America, all'interno del programma *"Very High Speed Integrated Circuit (VHSIC)"* iniziato nel 1980.

Questo linguaggio è stato creato con l'intenzione di soddisfare i bisogni che si riscontravano nel processo di progettazione. Le caratteristiche principali di questo linguaggio sono:

- la possibilità di rappresentare un sistema come composizione di sottosistemi più semplici e di descrivere come questi sottosistemi siano interconnessi fra loro;
- la possibilità di rappresentare il funzionamento del sistema utilizzando una forma di linguaggio di programmazione familiare;
- la possibilità di simulare il sistema progettato prima che venga effettivamente realizzato e confrontare così scelte progettuali diverse;
- la possibilità di sintetizzare i circuiti partendo sia dal livello comportamentale che da quello di porta logica.

2.1.3.2 SystemC

La crescente domanda di sistemi on chip (SoC) e soprattutto l'aumentare della densità dei circuiti integrati hanno aperto nuove sfide nella loro progettazione. Infatti la funzionalità e quindi il sistema implementabile in un singolo circuito integrato è sempre più complesso ed è quindi necessario che i progettisti lavorino ad un livello di astrazione maggiore per poter mantenere la produttività richiesta. Un altro aspetto che sta influenzando negli ultimi anni la progettazione di sistemi dedicati è la necessità sempre più presente di integrare fortemente le componenti hardware e software del sistema. Sono quindi necessari nuovi approcci per la specifica, il partizionamento e la verifica del sistema. L'utilizzo di linguaggi diversi, linguaggi di specifica Hardware come Verilog o VHDL per la componente HW e linguaggi di programmazione come C o C++ per la componente SW rende comunque difficoltosa questa integrazione.

Una possibile soluzione a questo problema fu individuata in una nuova piattaforma che fosse comune ai due rami di sviluppo di progetto (HW e SW) e che fosse basata su un linguaggio di specifica che consentisse il contemporaneo sviluppo e verifica delle funzionalità del sistema sia nelle sue componenti Hardware che in quelle Software. La scelta per tale linguaggio è caduta sul C++, opportunamente esteso, perchè solo questo linguaggio poteva garantire un adeguato livello di astrazione, assicurando al contempo una elevata integrazione tra parte hardware e parte software del sistema.

L'estensione proposta ha preso il nome di SystemC: essa è stata sviluppata come un linguaggio standardizzato di specifica e di modellizzazione in grado di poter essere utilizzato per descrizioni a diversi livelli di astrazione e per descrivere sistemi contenenti sia parti hardware che parti software. SystemC è interamente basato su C++, ed il codice relativo al simulatore SystemC è scaricabile gratuitamente da [Sys] con licenza OCL. Lo standard viene gestito e controllato da un gruppo di tredici componenti in parte di provenienza industriale e in parte accademica.

Fra le caratteristiche peculiari di SystemC si può ricordare:

- permette di descrivere il sistema anche ad un livello di astrazione maggiore rispetto a quello RT;

- permette di descrivere contemporaneamente componenti Hardware e Software;
- a livello RT i costrutti di modellazione forniti sono simili a quelli degli altri linguaggi di descrizione dello Hardware;
- offre costrutti per rappresentare facilmente numeri in virgola fissa;
- la concorrenza, similmente al VHDL, viene modellata attraverso l'uso di processi;
- introduce un insieme di funzionalità per una generica modellazione della comunicazione e della sincronizzazione.

I vantaggi offerti da questo linguaggio sono :

- la validazione del progetto può essere realizzata contemporaneamente sia per le componenti hardware che software;
- il testbench è unico e scritto in un linguaggio basato su C/C++ sia per lo hardware che per il software;
- le simulazioni possono essere fatte contemporaneamente su software e hardware e potendo essere fatte ad un livello di astrazione superiore all'RT sono generalmente più veloci.

La caratteristica comune di tutti i vantaggi è quella di poter operare a qualsiasi livello della sintesi di alto livello contemporaneamente su hardware e software.

Ci sono anche degli svantaggi che più che tecnologici sono di natura umana; infatti fino all'introduzione del SystemC le progettazioni di componenti HW e SW costituivano due attività diverse e abbastanza scorrelate. Pertanto permangono tuttora delle difficoltà nell'integrare questi due ambienti e giungere così alla diffusione della figura del progettista HW/SW: i progettisti Hardware hanno scarsa conoscenza del C++, i progettisti software hanno scarsa conoscenza degli strumenti e dei problemi relativi alla progettazione Hardware.

2.1.4 Rappresentazioni intermedie

Le rappresentazioni intermedie sono quelle rappresentazioni utilizzate nelle diverse fasi della Sintesi ad Alto Livello. Ogni fase può utilizzare un tipo di rappresentazione diverso per velocizzare la sua computazione, mettere in risalto alcune caratteristiche del sistema o semplificare la scrittura dei risultati intermedi.

La caratteristica che accomuna in generale i tipi di rappresentazione più usati è il fatto di essere costituiti da grafi orientati. In particolare i nodi di tali grafi solitamente rappresentano operazioni (o insieme di operazioni come nel caso degli ultimi due tipi di grafo che verranno a breve elencati) mentre gli archi rappresentano delle dipendenze di controllo o di dato fra di esse. Al fine della costruzione di alcune di queste rappresentazioni intermedie, in particolare di quelle che mostrano le dipendenze di controllo, sussiste una grande differenza fra le operazioni rappresentanti costrutti di controllo e operazioni rappresentanti computazioni. Invece, sempre al fine della costruzione dei grafi, distinguere fra i diversi tipi di operazioni computazionali non ha importanza, ma è comunque necessario memorizzarne le caratteristiche all'interno dei grafi stessi affinchè tali informazioni possano essere utilizzate dalle fasi della Sintesi al Alto Livello.

Le rappresentazioni basate su grafi orientati che verranno descritte sono:

- *Data Flow Graph* (DFG)
- *Control Flow Graph* (CFG)
- *Data Dependence Graph* (DDG)
- *Control Dependence Graph* (CDG)
- *System Dependence Graph* (SDG)
- *Control Data Flow Graph* (CDFG)
- *Grafo dei Blocchi Basici*

Fra queste rappresentazioni quelle che verrano utilizzate dall'implementazione dell'algoritmo realizzata sono il System Dependence Graph e il Grafo dei Blocchi Basici insieme ad un'ulteriore rappresentazione basata su grafi non orientati (l'albero dei dominatori) che verrà anch'essa successivamente descritta.

2.1.4.1 Data Flow Graph

Il Data Flow Graph (DFG) è un grafo in cui sono rappresentate le operazioni presenti in un sistema e le dipendenze dati esistenti tra di esse.

Formalmente il DFG è un grafo orientato $G_d(V, E)$ dove:

$V = \{v_i; i = 1, 2, ..., n_{op}\}$ è l'insieme dei nodi del grafo, dove n_{op} è il numero di operazioni, che si trovano in relazione uno a uno con le singole operazioni del sistema;

$E = \{(v_i, v_j); i, j = 1, 2, ..., n_{op}\}$ è l'insieme degli archi orientati del grafo: se esiste un arco tra v_i e v_j significa che esiste un trasferimento di dati tra l'i-esima operazione e la j-esima, ovvero che vi è una dipendenza di dato fra le due operazioni.

Nel modello DFG non vengono esplicitate le variabili necessarie alla memorizzazione dei dati trasferiti (a differenza del DDG - 2.1.4.3) fra i diversi nodi del grafo legati da dipendenza. Questo modello può inoltre venir utilizzato per la rappresentazione delle dipendenze dato fra sistemi composti da più processi per permettere una facile analisi e gestione del flusso di comunicazione fra di essi.

2.1.4.2 Control Flow Graph

Il Control Flow Graph (CFG) è un grafo in cui si rappresentano le operazioni in sequenza così come sono ordinate nella specifica evidenziando i possibili branch. Dal punto di vista formale il CFG è un grafo orientato e connesso $G_d(V, E)$ dove:

$V = \{v_i; i = 1, 2, ..., n_{op}\}$ è l'insieme dei nodi del grafo, dove n_{op} è il numero di operazioni, che si trovano in relazione uno a uno con le singole operazioni del sistema;

$E = \{(v_i, v_j); i, j = 1, 2, ..., n_{op}\}$ è l'insieme degli archi orientati del grafo: esiste un arco tra v_i e v_j se

- la j-esima operazione segue l'i-esima operazione nella specifica del sistema
- la j-esima operazione è una delle possibili destinazioni dell'i-esma operazione che è di branch, cioè un'operazione che può provocare la redirezione del flusso di controllo come cicli, salti, diramzioni, ecc. .

Questa rappresentazione ha il vantaggio di essere facilmente estraibile dalla specifica iniziale, ma non evidenzia il massimo parallelismo estraibile dal sistema. E' inoltre non adatta a rappresentare sistemi multiprocesso in quanto nel grafo non è immediatamente evidenziabile quali possibili combinazioni fra i diversi percorsi relativi ai diversi processi possono essere contemporaneamente attivi, ne è possibile modellizzare facilmente sincronizzazioni o fork.

2.1.4.3 Data Dependence Graph

Il Data Dependence Graph (DDG) è un grafo che similmente al DFG rappresenta le operazioni presenti in un sistema e le dipendenze dati esistenti tra di esse, ma a differenza del DFG evidenzia anche la variabile che è causa della dipendenza.

Formalmente il DDG è un grafo orientato $G_d(V, E)$ dove:

$V = \{v_i; i = 1, 2, ..., n_{op}\}$ è l'insieme dei nodi del grafo, dove n_{op} è il numero di operazioni, che si trovano in relazione uno a uno con le singole operazioni del sistema;

$E = \{(v_i, v_j); i, j = 1, 2, ..., n_{op}\}$ è l'insieme degli archi orientati del grafo: se esiste un arco tra v_i e v_j etichettato x_k devono verificarsi queste condizioni;

- l'operazione v_i contiene una definizione di x_k;
- l'operazione v_j contiene un uso di x_k;
- esiste un percorso p, cioè una possibile traccia di esecuzione, fra le operazioni v_i e v_j le cui operazioni non contengano definizioni di x_k.

2.1.4.4 Control Dependence Graph

Il CDG è un grafo che rappresenta le operazioni presenti in un sistema e le dipendenze di controllo fra di esse. Un'operazione x_j ha una dipendenza di controllo da x_i se x_i è un'operazione di branch e a seconda della direzione del salto intrapreso l'operazione x_j può essere eseguita o meno.

Formalmente il CDG è un grafo orientato $G_d(V, E)$ dove:

$V = \{v_i; i = 1, 2, ..., n_{op}\}$ è l'insieme dei nodi del grafo, dove n_{op} è il numero di operazioni, che si trovano in relazione uno a uno con le singole operazioni del sistema;

$E = \{(v_i, v_j); i, j = 1, 2, ..., n_{op}\}$ è l'insieme degli archi orientati del grafo: esiste un arco tra v_i e v_j se si verificano entrambe queste condizioni:

- esiste un percorso p da v_i a v_j, tale che v_j post-domina ogni nodo strettamente compreso tra v_i e v_j lungo il percorso p;
- v_j non post-domina il nodo v_i.

La definizione di post-dominanza è la seguente: un nodo v_i post-domina v_j se v_i è diverso da v_j e v_i fa parte di ogni percorso del CFG fra v_j e l'uscita.

2.1.4.5 System Dependence Graph

La rappresentazione SDG è il risultato della fusione di due rappresentazioni intermedie: il CDG e il DDG.

Formalmente il SDG è un grafo orientato $G_d(V, E)$ dove:

$V = \{v_i; i = 1, 2, ..., n_{op}\}$ è l'insieme dei nodi del grafo, dove n_{op} è il numero di operazioni, che si trovano in relazione uno a uno con le singole operazioni del sistema e quindi in corrispondenza uno a uno con i nodi del CDG e i nodi del DDG;

$E = \{(v_i, v_j); i, j = 1, 2, ..., n_{op}\}$ è l'insieme degli archi orientati del grafo: esiste un arco tra v_i e v_j se si verifica una di queste condizioni:

- esiste un arco tra v_i e v_j nel CDG;

– esiste un arco tra v_i e v_j nel DDG.

Questa rappresentazione è quella più utile per gli algoritmi di scheduling perchè permette di considerare contemporaneamente sia le dipendenze dato, sia le dipendenze di controllo e non preordina le operazioni secondo la sequenza in cui si trovano della descrizione della specifica.

2.1.4.6 Control Data Flow Graph

A differenza delle rappresentazioni precedenti in questo tipo di grafo non vi è una corrispondenza biunivoca con le operazioni della descrizione del sistema. Le sequenze massime di operazioni che non contengano al loro interno operazioni di salto condizionato ne contengono operazioni destinazione di salti condizionati (ad eccezione della prima operazione della sequenza) vengono raggruppate fra di loro e fatte corrispondere ad un nodo nel grafo. Più precisamente si fa corrispondere ad un nodo del CDFG non il mero elenco delle operazioni, ma il DFG parziale che descrive le relazioni di dipendenza dato fra di esse. Le operazioni che non fanno parte di alcune sequenza come per esempio le operazioni di salto condizionato vengono anch'esse fatte corrispondere ad un nodo del grafo.

Formalmente il CDFG è un grafo orientato $G_d(V, E)$ dove:

$V = \{v_i; i = 1, 2, ..., n\}$ è l'insieme dei nodi del grafo costruito come indicato;

$E = \{(v_i, v_j); i, j = 1, 2, ..., n_{op}\}$ è l'insieme degli archi orientati del grafo: esiste un arco tra v_i e v_j se

– la prima operazione del j-esimo nodo segue una delle operazioni dell'i-esimo nodo nella specifica del sistema;

– una qualche operazione del j-esimo nodo è una delle possibili destinazioni dell'operazione di salto condizionato corrispondente all'i-esimo nodo.

Nella sua struttura il CDFG è quindi simile al CFG con la differenza che le catene di operazioni di quest ultimo vengono inserite all'interno di un DFG parziale che costituirà un nodo singolo nel CDFG.

2.1.4.7 Grafo dei Blocchi Basici

Un blocco basico, così come definito in [App98], è una sequenza di operazioni che gode delle seguenti proprietà:

- la prima operazione è una destinazione di un salto;
- l'ultima operazione è un salto;
- non vi sono altre operazioni di salto o destinazione di essi (operazioni etichettate) all'interno della sequenza.

Pertanto se un'operazione di un blocco basico viene eseguita in una certa traccia, tutte le operazioni di quel blocco basico verranno eseguite in quella traccia. I blocchi basici di una descrizione di sistema possono a loro volta costituire i nodi di un grafo; tale grafo è molto simile al CDFG ad eccezione che, se un'operazione di salto ha un unico nodo cioè un unico blocco basico come predecessore diretto allora ad esso verrà assegnata e che le operazioni appartenenti ad un unico nodo non vengono descritte da un DFG parziale ma semplicemente elencate all'interno del nodo stesso così come compaiono nella specifica.

Formalmente il grafo dei Blocchi Basici è un grafo orientato $G_d(V, E)$ dove:

$V = \{v_i; i = 1, 2, ..., b\}$ è l'insieme dei nodi che si trovano in relazione uno a uno con i blocchi basici;

$E = \{(v_i, v_j); i, j = 1, 2, ..., b\}$ è l'insieme degli archi orientati del grafo: esiste un arco tra v_i e v_j se:

- la prima operazione del j-esimo blocco basico segue l'ultima operazione dell'i-esimo blocco basico;
- la prima operazione del j-esimo blocco basico è una delle possibili destinazioni del salto con cui termina l'i-esimo blocco basico.

2.1.4.8 Albero dei Dominatori

A differenza delle precedenti rappresentazioni, l'Albero dei Dominatori come dice il nome stesso non è un grafo orientato bensì un albero ed in particolare costituisce un'elaborazione del grafo dei Blocchi Basici.

Un blocco basico B_i domina un blocco basico B_j se ogni percorso nel grafo dei blocchi basici fra il blocco di ingresso e B_j passa da B_i. Un blocco basico può avere quindi più dominatori. La relazione di dominanza viene estesa includendo le seguenti dominanze:

- il blocco di ingresso domina tutti i nodi del grafo;
- ogni blocco domina se stesso.

Dalla relazione di dominanza si ricava la relazione di dominanza immediata, più utile per analizzare le caratteristiche di un sistema: un blocco basico B_i domina direttamente B_j se:

- B_i domina B_j;
- non esiste un B_k tali che B_i domina B_k e B_k domina B_j.

A differenza della relazione di dominanza semplice, ogni blocco basico ha un unico dominatore immediato. A partire dalla dominanza diretta viene costruito l'albero dei dominatori: la radice dell'albero è il blocco basico iniziale; i figli della radice sono i blocchi basici dominati direttamente dal blocco basico iniziale; i figli di un nodo qualsiasi sono i blocchi basici dominati direttamente dal blocco basico corrispondente a quel nodo.

2.2 Il problema dello scheduling

Il problema dello scheduling consiste nell'assegnamento delle operazioni che compongono il sistema ad uno specifico passo di controllo, cercando di minimizzare una prefissata funzione obiettivo e rispettando nel contempo un'insieme di vincoli. Funzione obiettivo e vincoli possono riguardare il tempo (numero di passi di controllo o ritardo del sistema) o la tecnologia (numero di unità funzionali o potenza dissipata).

Nell'ipotesi semplificatrice che tutte le operazioni abbiano tempo di esecuzione uguale (per soddisfare questa ipotesi è sufficiente considerare il tempo di esecuzione di ogni operazione pari al tempo di quella più lenta) e posta la durata di un passo di controllo pari a tale tempo, un'operazione può essere assegnata solo in un passo di controllo successivo a tutti quelli ai quali sono stati assegnati i suoi predecessori e precedente a quelli nei quali sono stati assegnati i suoi successori.

L'ipotesi appena fatta è tuttavia troppo conservativa e rischia di essere troppo penalizzante nel processo di minimizzazione del tempo di esecuzione della funzionalità: il tempo di clock ricavato potrebbe risultare infatti troppo lungo. E' quindi necessario considerare la possibilità che il ritardo introdotto nell'esecuzione delle singole operazioni possa essere diverso. Gli approcci a questa nuova problematica sono sostanzialmente quattro:

- Multicycling

  In questo caso la durata del passo di controllo viene dimensionata sull'operazione più veloce; l'esecuzione delle operazioni più lunghe richiedrà più di un ciclo di controllo (per questo motivo vengono definite operazioni multiciclo). Poichè i dati in ingresso dell'operazione multiciclo potrebbero essere sovrascritti dopo il primo ciclo di clock dall'inizio della sua esecuzione, è necessario introdurre dei latch all'ingresso dell'unità funzionale che la esegue affinchè i dati vengano tenuti memorizzati. Questo approccio ha tuttavia il difetto di aumentare il numero di passi di controllo richiesti e conseguentemente la complessità dell'unità di controllo.

- Chaining

  Il tempo di clock viene mantenuto uguale a quello dell'operazione più lenta, tuttavia si consente di eseguire due o più operazioni in serie nello stesso passo di controllo. Sostanzialmente si permette di assegnare un'operazione allo stesso ciclo al quale è assegnato l'ultimo dei suoi predecessori purchè la somma dei tempi di esecuzione delle operazioni che costituiscono la serie (operazioni concatenate) sia inferiore al periodo di clock. Tale approccio ha lo svantaggio di richiedere collegamenti diretti fra le diverse unità funzionali oltre a quelli già presenti fra registri ed unità.

- Multicycling + Chaining

  Il periodo di clock viene posto uguale ad un valore medio dei tempi di esecuzione delle operazioni o al tempo di esecuzione più diffuso fra le diverse operazioni. Le operazioni con tempo di esecuzione maggiore del periodo di clock verranno considerate come multiciclo e si permetterà di fare chaining fra quelle con tempo di esecuzione minore della durata di un passo di controllo.

- Pipeline

  Questo approccio è complementare a quelli fin qui presentati e richiede la presenza di unità funzionali apposite (unità dotate di pipeline). Le operazioni che possono essere eseguite da questo tipo di unità sono considerate al pari delle operazioni multiciclo, ma l'unità funzionale che le esegue viene considerata disponibile a partire dal passo di controllo successivo a quello in cui un'operazione le è stata assegnata.

Sommariamente gli algoritmi possono suddividersi in due categorie: quelli che cercano di minimizzare una metrica riguardante le temporizzazioni e quelli che cercano di minimizzare una metrica riguardante la tecnologia.

2.2.1 Algortimi che minimizzano metriche temporali

All'interno degli algoritmi che hanno come obiettivo quello di minimizzare il tempo di esecuzione (quantità che può essere espressa in modi diversi come per esempio ritardo introdotto dal sistema o numero di passi di controllo) si possono individuare due ulteriori sottocategorie:

- algoritmi non vincolati: ASAP, ALAP, Path-Based;

- algoritmi con vincoli sulle risorse: List-Based, Static-List.

  Questi algoritmi garantiscono di rispettare i vincoli sulle risorse che si concretizzano in un numero fissato di risorse allocate: a meno della presenza di istruzioni in mutua esclusione introdotta dall'esistenza di costrutti condizionali nel problema, non è possibile schedulare più operazioni di

un certo tipo rispetto al numero di unità funzionali allocate che possono eseguire quelle operazioni.

2.2.1.1 ASAP

Insieme all'ALAP è uno degli algoritmi più semplici: assegna ogni operazione al primo passo di controllo in cui questa può essere eseguita rispettando le dipendenze di dato e di controllo presenti nel CDFG. Ogni operazione viene schedulata appena è possibile (*As Soon As Possible*) come descritto in [CD86] e [CM87]:

1. si schedulano tutte le operazioni senza predecessori nel primo passo di controllo;
2. finchè vi sono operazioni ancora da schedulare si ripete:
   (a) si incrementa di uno il numero di passi di controllo totale;
   (b) si schedula nel passo di controllo appena creato una operazione se i suoi predecessori sono già tutti stati schedulati in passi di controllo precedenti;

La soluzione trovata è ottima: non esiste uno scheduling possibile che utilizzi un numero di passi di controllo inferiore a quello calcolato dall'ASAP.

2.2.1.2 ALAP

E' il duale dell'ASAP: assegna ogni operazione all'ultimo passo di controllo in cui questa può essere eseguita rispettando le dipendenze di dato e di controllo presenti nel CDFG. Ogni operazione viene schedulata il più tardi possibile (*As Late As Possible*):

1. si schedulano tutte le operazioni senza successori nell'ultimo passo di controllo;
2. finchè vi sono operazioni ancora da schedulare si ripete:
   (a) si decrementa di uno l'indice del passo di controllo corrente;

(b) si schedula nel passo di controllo corrente una operazione se i suoi successori sono già tutti stati schedulati in passi di controllo successivi.

La soluzione trovata è ottima: non esiste uno scheduling possibile che utilizzi un numero di passi di controllo inferiore a quello calcolato dall'ALAP e tale numero coincide ovviamente con quello calcolato dall'ASAP.

Relativamente ad una singola operazione, i passi di controllo compresi fra quello indicato dall'ASAP per quella operazione e quello indicato dall'ALAP prendono il nome di *finestra temporale* dell'operazione; il numero di tali passi di controllo, compresi gli estremi cioè quelli indicati da ASAP e ALAP prende il nome di mobilità dell'operazione.

2.2.1.3 Path–Based

Il *Path-Based scheduling* ([SA89] [Cam90] e [BP91]) minimizza il numero di passi di controllo necessari ad eseguire il percorso critico all'interno del CDFG. Il modo in cui procede l'algoritmo è il seguente: tutti i possibili percorsi di esecuzione sono estratti dal CDFG e sono schedulati indipendentemente l'uno dall'altro. Successivamente lo scheduling dei singoli percorsi viene combinato per generare la soluzione finale.

L'algoritmo genera dei vincoli tra nodi che vengono schedulati in passi di controllo differenti: Il problema di introdurre i vincoli relativi alla minimizzazione dei passi di controllo viene cioè trasformato in un problema di partizionamento di "clique" o di sottografi completamente connessi. In questo nuovo problema i nodi rappresentano i vincoli sugli intervalli mentre gli archi rappresentano i vincoli sulla sovrapposizione di intervalli. Una soluzione a questo partizionamento considererà la minima sovrapposizione di intervalli in un dato percorso. Gli intervalli dei differenti percorsi saranno poi combinati usando la stessa tecnica per ottenere un nuovo CDFG. L'introduzione di nuovi passi di controllo al CDFG finale produrrà infine una unica soluzione finale.

2.2.1.4 List Based

Come descritto in [DLSM81], il *List Based scheduling* fornisce una generalizzazione dell'algoritmo ASAP includendo vincoli sul numero delle risorse disponibili. Il funzionamento prevede che ad ogni passo di controllo:

1. si identifichino i nodi pronti ossia quei nodi i cui predecessori sono già stati schedulati;

2. si creino per ogni tipo di unità funzionale una lista di operazioni pronte che possano da esse essere eseguite; le liste devono essere ordinate in base ad una funzione di priorità;

3. per ogni lista di operazioni si schedulano le operazioni in ordine di priorità finche la lista è vuota o finchè non vi sono più unità funzionali di quel tipo disponibili; se vi sono più operazioni in lista rispetto al numero di unità funzionali si differisce lo scheduling di quelle con priorità minore.

Una possibile funzione di priorità può essere basata sulla mobilità delle operazioni (differenza fra ALAP e ASAP delle operazioni incrementata di un'unità): si può assegnare una priorità che è inversamente proporzionale alla mobilità. In questo modo in uno specifico passo di controllo i nodi con mobilità più bassa saranno quelli che avranno maggiore possibilità di essere schedulati anticipatamente. Una ridotta mobilità infatti implica che l'operazione possa essere schedulata in solo pochi passi di controllo. Un'altra possibile funzione di priorità è l'appartenenza o meno ad uno dei percorsi critici del CDFG.

I risultati dell'algoritmo dipendono fortemente dalla funzione di priorità scelta. La complessità temporale e spaziale per questa tipologia di algoritmo è abbastanza elevata, dal momento che è necessario mantenere dinamicamente un elevato numero di liste di priorità.

2.2.1.5 Static List

Questo algoritmo (proposto in [RH91]) è simile al List-Based, ma utilizza un'unica lista di priorità creata all'inizio dello scheduling che non viene aggiornata

dinamicamente. La base per il calcolo della priorità delle operazioni sono anche in questo caso i risultati forniti da ASAP e ALAP: le operazioni vengono ordinate secondo due chiavi diverse: la prima chiave di ordinamento è il passo di controllo assegnato dall'ALAP; la seconda chiave di ordinamento, che viene utilizzata per operazioni che hanno lo stesso ALAP, è il passo di controllo dell'ASAP. Per entrambe le chiavi si utilizza l'ordinamento decrescente; in caso di nodi con entrambe le chiavi uguali l'ordinamento è arbitrario. Si è così ottenuto un assegnamento assoluto di tutte le operazioni. A questo punto sulla base della posizione che le operazioni hanno nella lista si assegna loro una priorità: i nodi che appartengono all'inizio della lista saranno quelli a minore priorità, i nodi che appartengono alla coda saranno quelli a maggiore priorità.

Una volta stabilita la priorità delle operazioni è possibile eseguire lo scheduling: ad ogni passo di controllo si cerca di schedulare le operazioni a maggiore priorità purchè esistano unità funzionali libere capace di eseguirle e vengano rispettate le dipendenze. Qualora un'operazione non possa essere schedulata, il suo scheduling verrà posticipato. A differenza del List Based, la lista non viene aggiornata ad ogni iterazione (proprio da questo deriva l'aggettivo static). Il maggior vantaggio dell'algoritmo consiste proprio nel mancato aggiornamento in quanto in questo modo a differenza del List-Based è necessario costruire una lista di priorità delle operazioni un'unica volta.

2.2.2 Algortimi che minimizzano metriche architetturali

Gli algoritmi che hanno come obiettivo quello di minimizzare metriche riguardanti l'architettura (area occupata, potenza dissipata, numero di unità funzionali) sono caratterizzati dal dover avere il numero di passi di controllo fissato (ciò è fatto implicitamente nel caso del Simulated Annealing); questa caratteristica li rende molto importanti nella progettazione di sistemi real-time caratterizzati soprattutto dalla presenza di vincoli temporali e dalla necessità di ridurre aspetti come l'area o la potenza dissipata. Alcuni di questi algoritmi sono stati pensati per minimizzare solo una specifica metrica architetturale (per esempio il numero di unità funzionali), tuttavia poichè vi è forte dipendenza fra le diverse metriche è facile estenderli per ottenere l'ottimizzazione anche di

altre metriche da essa dipendenti. Esistono diversi approcci a questa classe di problemi:

- Matematico: Programmazione Lineare Intera (ILP);
- Stocastico: Simulated Annealing;
- Euristico: Force Directed;
- Raffinamento Iterativo: Rescheduling Iterativo.

2.2.2.1 Programmazione Lineare Intera

Il metodo della Programmazione Lineare Intera (ILP) ([JY89]) trova una soluzione ottima al problema della minimizzazione dei costi delle risorse attraverso una ricerca branch-and-bound che utilizza "backtracking" (2.3). Siano S_{Ek} e S_{Lk} i passi di controllo a cui è assegnata la k-esima operazione dagli algoritmi ASAP e ALAP. In uno scheduling che rispetti tutte le dipendenze l'operazione k non può essere schedulata prima di S_{Ek} e dopo S_{Lk}. Da questo, come è già stato mostrato, si ricava il valore della finestra temporale di una operazione che per definizione è pari a

$$M = \{S_j | E_k \leq j \leq L_k\},$$

Dalla finestra temporale di un'operazione si ricava facilmente la sua mobilità che è il numero di passi di controllo compresi tra S_{Ek} e S_{Lk} estremi inclusi. La notazione usata per questo tipo di formulazione prevede l'introduzione e la definizione delle seguenti quantità:

$OP = \{O_i | 1 \leq i \leq n\}$ insieme delle operazioni;

$t_i =$ tipo dell'i-esima operazione;

$T = \{t_k | 1 \leq k \leq m\}$ insieme dei possibili tipi;

OP_{tk} operazioni in OP di tipo t_k;

$INDEX_{tk}$ insieme degli indici di operazioni in OP_{tk};

N_{tk} numero di unità che compiono operazioni di tipo t_k;

C_{tk} costo di tali unità;

$S = \{s_j | 1 \leq j \leq r\}$ insieme di passi di controllo disponibili per lo scheduling delle operazioni;

x_{ij} variabili logiche che indicano l'assegnamento di un'operazione i ad un passo di controllo j: 1 in caso di assegnamento, 0 in caso contrario

Il problema dello scheduling può quindi essere formulato come:

$$minimizza(\Sigma_{k=1}^{m}(C_{tk} * N_{tk}))$$

con i vincoli:

- un'operazione può essere schedulata in un unico passo di controllo e questo passo di controllo deve far parte della sua finestra temporale

$$\forall i, 1 \leq i \leq n, (\Sigma_{E_j \leq j \leq L_j} x_{ij} = 1);$$

- il numero di operazioni di tipo t_k schedulate in un passo di controllo non può essere maggiore del numero di unità funzionali che possono eseguire operazioni di tipo t_k (nel caso non esistano operazioni in mutua esclusione a causa della presenza di costrutti di controllo)

$$\forall j, 1 \leq j \leq r, \forall k, 1 \leq k \leq m, (\Sigma_{j \in INDEX_{tk}} x_{ij} \leq N_{tk});$$

- un'operazione non può essere schedulata prima dei successori

$$\forall i, j, o_i \in Pred_{o_j}(\Sigma_{E_i \leq k \leq L_i}(k \cdot x_{ik}) - \Sigma_{E_j \leq l \leq L_j}(l \cdot x_{jl}) \leq -1).$$

L'algoritmo ha complessità elevata e quindi il tempo di esecuzione cresce molto rapidamente al crescere del numero di variabili utilizzate. Non è quindi utilizzabile direttamente in casi che non siano molto semplici: ne sono state pertanto

realizzate versioni modificate che non fanno uso di backtracking, ma utilizzando tecniche euristiche assegnano un'operazione alla volta invece di tentare di trovare una soluzione completa.

2.2.2.2 Simulated Annealing

L'algoritmo *Simulated Annealing* è descritto in [SA89]. Il problema dello scheduling in questo caso viene rappresentato attraverso una tabella in cui le righe raffigurano i passi di controllo, mentre le colonne rappresentano le unità funzionali disponibili: il problema è quindi quello di posizionare le operazioni all'interno di questa tabella, imponendo il vincolo legato al fatto che in un passo di controllo una risorsa può essere utilizzata per un'unica operazione. Ogni cella conterrà quindi una sola operazione (considerando solo problemi senza costrutti condizionali).

L'algoritmo parte da una soluzione iniziale che deve essere precalcolata e per successive iterazioni effettua delle modifiche allo scheduling iniziale. Ad ogni iterazione viene valutata la bontà di questa modifica, che può essere accettata con una certa probabilità dipendente dal guadagno ottenuto relativamente alla funzione obiettivo. La funzione obiettivo tipicamente riguarda aspetti architetturali del sistema. Anche in caso di peggioramento delle prestazioni o della funzione obiettivo è possibile, seppur ovviamente con probabilità minore, che la nuova soluzione venga accettata. L'accettare anche soluzioni peggiorative permette di uscire dai minimi locali della funzione obiettivo garantendo in tal modo una più esaustiva esplorazione dello spazio delle soluzioni.

Per tale motivo questo algoritmo può essere considerato particolarmente robusto, anche se tale robustezza si ripercuote in termini di tempi di esecuzione.

2.2.2.3 Force Directed

Il *Force Directed* scheduling ([PK87] e [PK89]) è l'algoritmo di cui si propone un'estensione in questo lavoro di tesi, quindi verrà descritto accuratamente nel capitolo 3. Se ne dà qui per completezza del confronto con gli altri algoritmi presentati una breve descrizione.

Il Force Directed è un metodo euristico con l'obiettivo di minimizzare il numero totale di unità funzionali utilizzate fissato il numero di passi di controllo. Per minimizzre questo valore l'algoritmo cerca di distribuire uniformemente le operazioni di uno stesso tipo nei diversi passi di controllo. Come gli algoritmi precedenti necessita delle informazioni fornite da ASAP e ALAP.

La soluzione dell'algoritmo non è ottima a causa di alcune approssimazioni che vengono fatte durante la sua esecuzione sugli effetti prodotti dai singoli assegnamenti.

2.2.2.4 Rescheduling Iterativo

Il metodo del *Rescheduling Iterativo (IR)* ([ICCM91]) si basa sul problema della bisezione di grafi (proposto da Kernighan e Lin in [KS70]) e procede similmente al Simulated Annealing reschedulando singolarmente le operazioni e necessita quindi di una soluzione ammissibile già esistente. Tale soluzione viene modificata in questo modo: si sceglie un'operazione casualmente, la si sposta in un altro passo di controllo e la si blocca; si ripete ciò finchè tutte le operazioni sono bloccate. Dopo ogni modifica si valuta il guadagno che questa comporta: la sequenza di spostamenti che produce il miglior guadagno viene effettivamente applicata allo scheduling ottenendo una nuova soluzione. A partire da questa soluzione si riapplica una nuova iterazione dell'algoritmo finchè non si giunge ad una soluzione accettabile.

La bontà dei risultati dipende fortemente dalla soluzione iniziale utilizzata; esistono delle varianti che si possono introdurre per ovviare a questa limitazione rendendo meno critica la scelta della soluzione iniziale e migliorando la qualità media dei risultati; i miglioramenti che possono essere apportati sono:

- utilizzare diverse configurazioni iniziali: l'algoritmo è computazionalmente efficiente e quindi può essere applicato più volte;

- utilizzare una migliore strategia di previsione degli effetti.

2.3 La tecnica del Backtracking

Il *Backtracking* è una tecnica di tipo costruttivo utilizzata nel risolvere problemi caratterizzati dall'esistenza di vincoli da soddisfare. Il termine *backtrack* fu coniato dal matematico americano D. H. Lehmer nel 1950, ma la prima generale esposizione di questa tecnica è dovuta a Walker ([Wal60]) e fra i primi studi generali si possono ricordare [GB65] e [BeMR75]. I problemi che questa tecnica può risolvere sono modellizzabili solitamente come la scelta di quale valore assegnare ad una serie di variabili. Una possibile rappresentazione di questo modello sono gli alberi di ricerca: ai nodi intermedi corrispondono soluzioni intermedie, alle foglie le soluzioni complete, agli archi le possibili scelte di assegnamento di una variabile e quindi i passi che contribuiscono a costruire la soluzione finale.

La ricerca della soluzione può essere modellizzata come una visita deptu-first dell'albero di ricerca (una volta giunti in un nodo la visita procedeh nel primo dei suoi figli; solo nel caso tutti i figli di un nodo siano già stati visitati si passa alla visita dei suoi fratelli). Per ottenere la soluzione ottima, cioè la soluzione in cui il valore di una certa funzione obiettivo è minimo, è necessario esplorare l'intero albero.

La tecnica appena descritta impone quindi di analizzare tutte le soluzioni possibili per individuare quella ottima. Per limitare il numero di soluzioni da esaminare è possibile utilizzare il metodo del *Branch and Bound*; questo consiste nel calcolare il soddisfacimento dei vincoli e il valore della funzione obiettivo non solo per le foglie, ma per tutti i nodi dell'albero e tagliare i sottoalberi che offrono soluzioni sicuramente peggiori di quella trovata fino a quel momento. Si supponga per esempio che i contributi dei singoli assegnamenti alla funzione obiettivo siano tutti non negativi: se in un nodo il valore della funzione obiettivo è più grande di quello della miglior soluzione trovata, tutte quelle corrispondenti alle foglie appartenenti al sottoalbero avente origine in quel nodo avranno un valore della funzione obiettivo peggiore della soluzione temporanea e pertanto potranno essere ignorate, riducendo quindi il numero di soluzioni da considerare.

Il backtracking ha complessità esponenziale quindi è poco efficiente per risolvere problemi che non siano NP-completi; è possibile tuttavia integrare al

suo interno euristiche, come quella appena presentata del Branch and Bound, che, pur non riducendo la complessità del caso pessimo, possono ridurre il tempo di esecuzione medio.

2.3.1 Tecniche di Look-Ahead

Le tecniche di Look-Ahead sono tecniche che estendono ed integrano quella del Backtracking consentendo di ridurre il numero di nodi da visitare dell'albero sia nel caso l'obiettivo sia individuare una soluzione valida, sia nel caso si voglia trovare una soluzione il cui valore della funzione obiettivo sia soddisfacente. Nella versione classica del Backtracking la scelta del prossimo arco e quindi del prossimo assegnamento a partire da un certo nodo è casuale o determinata dall'ordinamento lessicografico degli assegnamenti stessi.

L'idea alla base del Look-Ahead è quella di stimare gli effetti dovuti al prossimo assegnamento prima di scegliere quale arco percorrere. In questo modo è possibile escludere assegnamenti che provocherebbero una violazione dei vincoli del problema e cercare di percorrere anticipatamente il ramo che porta ad una soluzione se non ottima almeno accettabile secondo dei criteri prefissati.

E' possibile estendere l'orizzonte visivo di questa tecnica a più passi cioè cercare di stimare la qualità delle soluzioni parziali costruite a partire da quella attuale aggiungendo non uno ma più assegnamenti. Il risultato è di ridurre il numero medio di backtracking necessari (nel caso si voglia individuare una soluzione ammissibile) oppure di ottenere una soluzione mediamente migliore (nel caso si stia cercando di minimizzare una certa metrica senza comunque voler ottenere la soluzione ottima) a prezzo di una crescita del tempo di computazione medio dovuta all'overhead necessario per calcolare le stime delle soluzioni parziali.

Capitolo 3

Il Force Directed Scheduling proposto da Paulin e Knight

L'algoritmo del Force Directed scheduling fu presentato per la prima volta da Paulin e Knight in [PK87] come nuovo componente del sistema ad approccio multiplo per la sintesi automatica denominato *HAL* ([PKG86]) e successivamente esteso in [PK89]. Nella sua formulazione base il Force Directed è un algoritmo di scheduling a tempo vincolato (i vincoli temporali possono essere requisiti di progetto oppure essere calcolati tramite altri algoritmi come ASAP (2.2.1.1), ALAP (2.2.1.2) o List-Based (2.2.1.4)) che mira a minimizzare il numero di unità funzionali, bus e registri richiesti per implementare una certa funzionalità tramite il bilanciamento in ogni passo di controllo del numero di operazioni assegnate ad un certo tipo di risorsa. Il bilanciamento della distribuzione delle operazioni induce ovviamente l'uniformarsi dell'utilizzo delle unità funzionali nei diversi passi di controllo e conseguentemente riduce il picco massimo di richieste di un certo tipo di risorsa. Tale numero corrisponderà al numero di unità funzionali di quel tipo che sarà necessario allocare per implementare la funzionalità. L'algoritmo è iterativo e costruttivo: ad ogni iterazione un'operazione, sulla base delle procedure che verranno esposte in 3.1, verrà assegnata ad un passo di controllo.

Nella prima sezione verrà illustrata la prima versione dell'algoritmo che prevede la semplificazione che tutte le operazioni vengano eseguite in un tempo

uguale e fissato che verrà identificato come durata del passo di controllo.

Nella seconda sezione verrà invece mostrato come è possibile superare queste limitazioni e come introdurre informazioni relative alla tecnologia target all'interno dell'algoritmo.

3.1 L'algoritmo nella sua versione base

L'algoritmo del Force Directed scheduling si fonda sul concetto di forza da cui prende il nome. Ad ogni coppia <operazione-passo di controllo> viene associata una forza cioè un numero reale indice dell'effetto del scegliere quel particolare assegnamento <operazione-passo di controllo> sulla uniformità della distribuzione delle operazioni di qualsiasi tipo non solo in quel passo di controllo, ma in tutti i passi di controllo influenzati. Un valore positivo indica un aumento delle operazioni concorrenti per le stesse risorse e quindi un peggioramento della situazione rispetto a quella desiderata, un valore negativo invece indica una diminuzione della concorrenza e quindi un possibile ridursi delle unità funzionali richieste.

Il metodo per calcolare questo valore prende spunto dalla legge di Hooke per i corpi elastici (molle):

$$F = -kx \tag{3.1}$$

dove:

F è la forza esercitata dalla molla che si oppone all'allungamento;

k è la costante di Hooke;

x è la distanza della quale viene allungata la molla.

L'idea è che venga assimilata ad una molla la somma delle probabilità delle operazioni relativa ad una coppia <passo di controllo-tipo di unità funzionale> cioè un numero non necessariamente intero che indichi il numero di operazioni di quel tipo che probabilmente verranno associate a quel passo di controllo (per rispettare i segni delle equazioni è più corretto dire che alla molla corrisponde il complemento rispetto ad un'ipotetica costante della somma di probabilità;

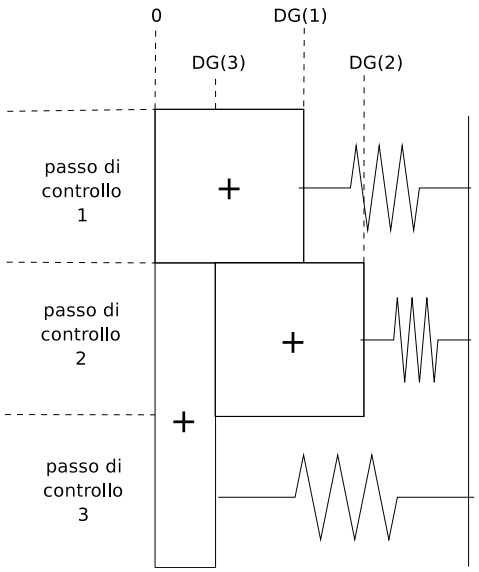


Figura 3.1. Esempio di interpretazione dell'algoritmo del Force Directed scheduling: i due quadrati ed il rettangolo contenenti il simbolo della somma rappresentano delle operazioni e la loro probabilità che siano assegnate ad un passo di controllo (maggiore è l'area, maggiore è la probabilità). La distanza fra 0 e DG(i) rappresenta la somma di probabilità per l'i-esimo passo di controllo: si ipotizza che le molle rappresentate nella parte destra abbiano costante di elasticità pari alle somme di probabilità dei rispettivi passi di controllo. Si ipotizza inoltre che tali molle si trovino in stato di compressione. Simbolicamente quindi le molle premono contro le operazioni cercando di uniformare il livello delle somme di probabilità per giungere ad uno stato stabile del sistema in cui le forze esercitate dalle singole molle siano uguali. Infatti la forza esercitata da una molla è tanto maggiore quanto maggiore è la somma di probabilità perchè maggiore è la sua costante elastica e la sua compressione. C'è da sottolineare tuttavia come a differenza di quelle reali, le molle utilizzate in questo esempio abbiano costante elastica variabile.

come calcolare le funzioni di distribuzione della probabilità delle operazioni che sono necessarie per calcolare la somma e come ottenere la somma stessa verrà illustrato in 3.1.1. Questa quantità, proseguendo nell'analogia con i corpi elastici (confrontare figura 3.1), esercita una forza nei confronti delle operazioni stesse che contribuiscono a formarla: la scelta di un assegnamento di un'operazione ad un passo di controllo comporterà la sua eliminazione da tutti i passi di controllo (con conseguente allungamento della molla e quindi insorgere di una forza negativa) tranne quello scelto in cui la somma di probabilità aumenterà (equivalentemente ad una compressione della molla e quindi ad una forza positiva). Sommando le forze dovute alle variazioni nei diversi passi di controllo si otterrà quindi un indice complessivo della variazione della concorrenza delle operazioni nel sistema. Una forza complessivamente negativa corrisponderà ad un allungamento medio delle molle e quindi alla diminuzione media delle somme di probabilità e quindi del numero medio di risorse necessarie nei diversi passi di controllo.

3.1.1 Calcolo delle somme di probabilità

Alla base del calcolo delle somme di probabilità c'è la necessità di conoscere la distribuzione di probabilità nei vari passi di controllo relativa ad ogni operazione. Per ottenerle si consideri innanzitutto che non tutte le operazioni sono schedulabili in tutti i passi di controllo. Le dipendenze di dato e le dipendenze di controllo impongono che alcune operazioni possano venir eseguite solo a condizione che quelle da cui esse dipendono siano concluse (a meno di considerare ipotesi speculative). Nei passi di controllo in cui è impossibile assegnare un'operazione non esistendo alcuno scheduling comprendente quell'assegnamento che possa soddisfare tutti i vincoli di dipendenza poniamo pari a zero la probabilità che quella operazione vi venga schedulata. Poichè a priori non disponiamo di alcuna altra informazione sulle probabilità nei diversi passi di controllo è bene utilizzare come distribuzione di probabilità quella che statisticamente minimizza l'errore di approssimazione nel caso di nessuna informazione: la distribuzione uniforme. Detto mobilità il numero di passi di con-

trollo in cui è possibile schedulare un'operazione, alla sua probabilità di venir assegnata in uno particolare di questi passi verrà attribuito il valore di $\frac{1}{mobilita'}$.

Rimane il problema di individuare in quali passi di controllo è possibile schedulare una certa operazione (denominati finestra temporale di un'operazione) e quindi la mobilità. La soluzione a ciò è offerta dagli algoritmi ASAP e ALAP: il primo fornisce infatti il primo passo di controllo in cui una certa operazione può essere schedulata, il secondo fornisce l'ultimo passo. Inoltre questi algoritmi calcolano in modo ottimale il numero di passi di controllo minimo necessario per eseguire tutte le operazioni rispettando i vincoli di dipendenza. Questo valore è anche quello che assume per il suo calcolo l'algoritmo del Force Directed scheduling base.

Una volta ottenuta le probabilità che le singole operazioni possano essere schedulate in un certo passo di controllo è necessario ricavare la somma di questi valori. Queste somme danno un indice del numero di unità funzionali che in quel dato di controllo probabilmente saranno utilizzate. Al di là dell'aspetto intuitivo di questa corrispondenza, viene qui presentata una giustificazione matematica non presente in [PK87] e [PK89]. Considerato un passo di controllo fissato e un tipo di operazione fissato, per ognuna delle operazioni di quel tipo schedulabili in quel passo di controllo viene creata una variabile aleatoria discreta X_i che può assumere valore 0 (l'operazione non viene schedulata in quel passo di controllo) o 1 (l'operazione viene assegnata a quel passo di controllo). Tale variabile casuale sarà evidentemente una Bernoulliana di parametro pari alla probabilità che tale operazione venga schedulata in quel passo. Al fine dei calcoli si può ipotizzare che le variabili siano tutte indipendenti (semplificazione insita nell'algoritmo stesso, ma non corrispondente alla realtà in quanto le probabilità che un'operazione venga schedulata in un certo passo dipende sì solo dalla sua mobilità, ma potrebbero esistere dipendenze fra le mobilità delle operazioni schedulabili contemporaneamente: per esempio un'operazione potrebbe dipendere dall'altra). Si consideri infine un'ulteriore variabile aleatoria Z che modellizzi il numero di operazioni del tipo fissato schedulate in quel passo di controllo. Si avrà quindi (nel caso di assenza di

costrutti condizionali):

$$Z = \sum_i X_i \tag{3.2}$$

con gli X_i che possono assumere valore 0 o 1. La media di questa variabile sarà

$$\mathbb{E}[Z] = \mathbb{E}[\sum_i X_i]$$

essendo le variabili indipendenti si può invertire l'operatore $\sum$ con $\mathbb{E}$

$$\mathbb{E}[Z] = \sum_i \mathbb{E}[X_i]$$

ma $\mathbb{E}[X_i] = p_i$ dove p_i è il parametro della bernouliana cioè la probabilità dell'operazione

$$\mathbb{E}[Z] = \sum_i p_i = DG(c)$$

Con DG(c) si indica la somma di probabilità nel passo *c* delle operazioni del tipo fissato.

In questo modo è stata dimostrata la stretta corrispondenza fra le somme di probabilità e l'utilizzo di un determinato tipo di unità funzionale. La somma di probabilità non può limitarsi però ad una mera somma algebrica proprio perchè il risultato finale dovrà essere un indice di quante unità funzionali di un certo tipo verranno probabilmente utilizzate in un determinato passo di controllo. Nel caso la funzionalità originaria che si voglia schedulare presenti al suo interno dei costrutti di controllo (come ad esempio degli IF, WHILE, etc.) è non solo possibile ma probabile che esistano una o più coppie (o combinazioni di numero maggiore) di operazioni in mutua esclusione reciproca, tali cioè che solo una di esse debba essere eseguita all'interno di una singola traccia di esecuzione della funzionalità. Pertanto è impossibile che esse richiedano simultaneamente l'utilizzo di una particolare unità funzionale, quindi per calcolare il numero di unità funzionali probabilmente occupate non è significativo sommare le probabilità delle singole operazioni, ma è necessario tenere in considerazione queste pos-

sibilità. Una prima approssimazione suggerirebbe di utilizzare la media delle probabilità delle operazioni in mutua esclusione, ma poichè lo scopo finale dell'algoritmo è comunque quello di minimizzare il massimo utilizzo di una certa unità funzionale, viene preso in esame il valore massimo fra di esse. Queste considerazioni relative a singole operazioni si estendono immediatamente in caso di situazioni più complesse con presenza di basic block in mutua esclusione. In questo caso si calcola la somma di probabilità relativa ai singoli blocchi basici sommando le probabilità delle operazioni appartenenti ad essi come se il blocco basico rappresentasse una macro-operazione ottenuta fondendo le singole operazioni. Tale macro-operazione potrà avere probabilità anche maggiore di uno potendo occupare più unità funzionali poichè è il risultato della fusione di diverse operazioni elementari. Una volta eliminati ai fini del calcolo i blocchi basici dominati a livello di probabilità da blocchi basici con probabilità totale maggiore è possibile ottenere il valore finale desiderato sommando le probabilità delle operazioni rimaste. Gruppi di operazioni appartenenti a rami di costrutti condizionali, ma che non siano in mutua esclusione con alcuna altra operazione dello stesso tipo vanno considerati al fine del calcolo come eseguite in ogni traccia di esecuzione.

3.1.2 Calcolo delle Forze

Una volta calcolate le somme delle probabilità è possibile calcolare le forze che stanno alla base dell'algoritmo di scheduling. Si calcola una forza per ogni coppia <operazione-passo di controllo possibile> (gli scheduling possibili come è stato evidenziato sono quelli calcolati dagli algoritmi ASAP - ALAP). Il valore di ogni forza si calcola sommando due diversi contributi, il primo detto *self-force* (autoforza) che si riferisce agli effetti dello scheduling dell'operazione in oggetto, il secondo detto *predecessors' and successors' force* (forza di predecessori e successori) che tiene conto della restrizione della finestre temporali dei predecessori (predecessori nel SDG cioè quelle operazioni da cui quella considerata dipende) e dei successori (successori nel SDG cioè quelle operazioni che dipendono da quella considerata) dell'operazione esaminata. Infatti uno scheduling equivale a restringere ad uno la mobilità di un'operazione, riduzione che può

causare la restrizione della mobilità di altre operazioni. Un assegnamento quindi può causare non solo la modifica delle somme di probabilità relative all'unità funzionale che esegue quella operazione e relative a quei passi di controllo interessati dall'operazione, ma anche di altri passi di controllo e di altre unità funzionali. Come già anticipato la funzione utilizzata per calcolare le forze ricalca la legge di Hooke:

$$Force(i) = DG(i) \cdot x(i) \tag{3.3}$$

dove

Force(i) è il contributo alla forza dello scheduling relativo all'i-esimo passo di controllo;

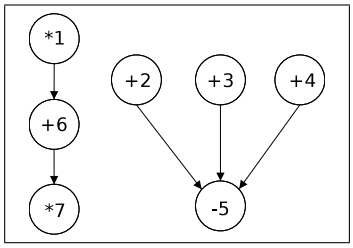
DG(i) è la somma di probabilità del tipo di unità funzionale che può eseguire l'operazione nell'i-esimo passo di controllo;

x(i) è la variazione della probabilità dell'operazione nell'i-esimo passo di controllo a seguito dello scheduling.

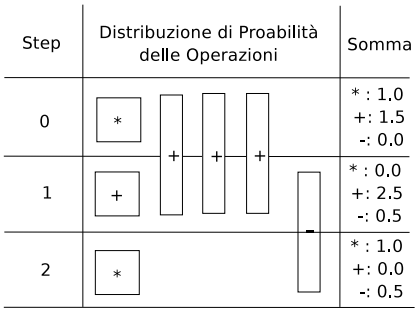
Questa formula permette di calcolare il contributo alla forza relativo ad un passo di controllo dell'assegnamento. La *self-force* complessiva è pari alla somma dei contributi relativi ai singoli passi di controllo. Ovviamente tali contribuiti saranno pari a zero per i passi di controllo nei quali non è possibile schedulare l'operazione perchè il terzo termine della formula sarà nullo. La variazione della probabilità dell'operazione è facilmente calcolabile in $-\frac{1}{mobilita'}$ per tutti i passi di controllo dove l'istruzione era schedulabile ad eccezione di quello dell'assegnamento che sarà pari a $+\frac{mobilita'-1}{mobilita'}$. Una possibile diversa interpretazione del terzo termine della formula e quindi del calcolo della forza verrà illustrata in 4.1. La *self-force* complessiva sarà positiva nel caso il passo di controllo dell'assegnamento abbia somma di probabilità mediamente superiore alle altre, negativa in caso contrario.

Allo stesso modo è possibile calcolare la *predecessors' and successors' force*: per ogni predecessore e successore la cui mobilità è modificata dal possibile scheduling si applica la formula utilizzata per il calcolo della *self-force* tenendo in considerazione le variazioni delle probabilità delle operazioni dovute alla restrizione della mobilità. A differenza del caso del calcolo della *self-force* la nuova

mobilità di un'operazione può essere maggiore di uno, quindi la formula per la modifica della probabilità illustrata precedentemente può non valere. Le singole forze così ottenute vengono sommate per dare origine alla *predecessors' and successors' force*. Va sottolineato come il calcolo delle forze di predecessori e successori avvenga singolarmente e non considerando complessivamente gli effetti della restrizione della mobilità di istruzioni contemporanee. Facendo riferimento all'analogia con la fisica descritta in 3.1 è come se si considerasse valido nel problema del calcolo delle forze il principio di sovrapposizione degli effetti. In realtà questa è solo una semplificazione introdotta dall'algoritmo perchè non corrisponde alla situazione reale del problema. Infatti bisognerebbe considerare che, dato lo scheduling di un'operazione avente più predecessori che subiscono una restrizione della mobilità a seguito dell'assegnamento stesso, si può notare nell'esempio riportato nella figura 3.2 come per calcolare la forza relativa a ciascun predecessore sarebbe più corretto considerare l'effetto causato dalla restrizione della mobilità degli altri predecessori a lui contemporanei in quanto questa restrizione modifica la somma di probabilità e quindi anche la forza. Infatti se si analizza la tabella riportata in 3.2(c) riferendosi in particolare alle forze dell'operazione *-5* si può notare come la forza relativa allo scheduling nel passo di controllo 1 sia nettamente negativa. Infatti questo scheduling, la cui *self-force* è 0 perchè i due passi in cui è possibile schedulare l'operazione hanno la stessa somma di probabilità, comporta la riduzione del frame delle operazione *+2*, *+3* e *+4* da [0 1] a [0 0] con un contributo di forza negativo poichè la somma di probabilità iniziale relativa all'operazione di somma nel passo 1 è superiore a quella del passo 0. Apparentemente quindi lo scheduling spinge queste tre operazioni in un passo di controllo mediamente meno congestionato come ci si aspetta che faccia l'algoritmo. Tuttavia è facile notare che lo scheduling contemporaneo delle tre istruzioni al primo passo di controllo non costituisce la soluzione ottimale al problema di scheduling presentato. Confrontando lo scheduling ottenuto dall'applicazione dell'algoritmo in 3.2(d) con quello costruito manualmente in 3.2(e) si può notare come la soluzione ottimale utilizzi due sommatori contro i tre calcolati dal Force Directed. Un possibile modo per ovviare a questa limitazione della formulazione originaria dell'algoritmo verrà presentato in 4.1.3.



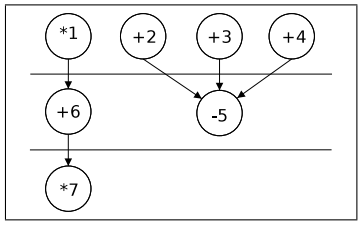
(a) DFG di esempio



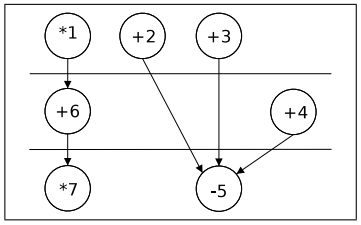
(b) Distribuzione di probabilità e somma di probabilità

Op	CS	SF	OF	TF
+2	0	-0.50	0	-0.50
+2	1	+0.50	0	+0.50
+3	0	-0.50	0	-0.50
+3	1	+0.50	0	+0.50
+4	0	-0.50	0	-0.50
+4	1	+0.50	0	+0.50
-5	1	0	-1.50	-1.50
-5	2	0	0	-0

(c) Tabella delle forze delle operazioni da schedulare
SF = self force
OF = predecessors' and successors' force
TF = total force



(d) Scheduling calcolato dall'Algoritmo Force Directed



(e) Scheduling con risorse minime

Figura 3.2. Esempio delle limitazioni insite nel calcolo delle *successors' and predecessors' forces*

3.1.3 Corpo dell'algoritmo

Dopo aver mostrato le singole fasi dell'algoritmo, è ora possibile illustrarlo nella sua completezza. L'algoritmo è iterativo ed ad ogni iterazione un'operazione viene assegnata ad un passo di controllo. Il numero massimo di iterazioni quindi è pari al numero di operazioni, ma può essere minore perchè è possibile assegnare in qualsiasi momento un'operazione che abbia mobilità unitaria.

I passi da compiere ad ogni iterazione sono:

1. calcolare la finestra temporale delle operazioni tramite l'utilizzo di ASAP e ALAP e derivare da questa la loro mobilità e quindi le distribuzioni di probabilità;

2. aggiornare le somme di probabilità;

3. calcolare le *self-forces* per ogni assegnamento <operazione-passo di controllo> possibile;

4. aggiungere alle *self-forces* le forze di predecessori e successori;

5. scegliere la coppia <operazione-passo di controllo> avente forza minore: aggiungere l'assegnamento allo scheduling.

3.1.4 Considerazioni sulla complessità

Viene ora analizzata la complessità dell'algoritmo, considerando il caso pessimo, ricalcando quanto riportato in [PK87] con alcune considerazioni ulteriori; con n è indicato il numero di operazioni del problema, con c il numero di passi di controllo previsto:

1. ad ogni iterazione dell'algoritmo almeno un'operazione viene schedulata (tipicamente più di una, ma si sta considerando il caso peggiore); il numero massimo di iterazioni dell'algoritmo è quindi pari ad n;

2. ad ogni iterazione vengono calcolate le forze di massimo n operazioni;

3. per ognuna delle operazioni si devono calcolare h forze dove h è la mobilità delle operazioni; nel caso peggiore la mobilità corrisponde al numero totale di passi di controllo quindi $h = c$;

4. Per calcolare una forza è necessario calcolare le forze relative a tutti i predecessori e successori che nel caso peggiore sono pari a *n-1*; una considerazione non presente nell'analisi formulata da Paulin e Knight: per calcolare la forza relativa ad un predecessore o un successore si devono sommare i contribuiti relativi a ciascuno dei passi di controllo in cui l'operazione è schedulabile; quindi ricalcando le considerazione fatte nel passo precedente la complessità va aumentata di un ulteriore fattore pari a c.

Da queste considerazioni si ricava che la complessità dell'algoritmo è pari a $O(c^2n^3)$. Tuttavia in questi calcoli non è stata considerata la possibilità dell'esistenza di costrutti condizionali. In questo caso come illustrato in 3.1.1 è necessario calcolare le somme di probabilità considerando le reciproche mutue esclusioni. Questi calcoli devono essere ripetuti ad ogni iterazione per ogni passo di controllo. Affinchè il tempo per individuare il valore di queste somme sia trascurabile al fine di calcolare la complessità dell'algoritmo è necessario che tali calcoli abbiano complessità inferiore a $O(cn^2)$, tuttavia ne in [PK87] ne in [PK89] è riportato un metodo rapido con cui effettuarli. Un metodo verrà illustrato in 5.2.3 avente complessità pari a $O(b^4)$ ove b è il numero di blocchi basici del problema.

Esistono delle semplificazioni all'algoritmo che comportano una riduzione della complessità:

- limitare la mobilità delle istruzioni ad un valore prefissato, ad esempio a 10 passi di controllo; la finestra temporale sarà centrata sul centro della finestra originaria; il tempo necessario ad effettuare la riduzione, che deve essere effettuata ad ogni iterazione, è $O(cn)$ quindi trascurabile rispetto alla complessità del corpo dell'iterazione; nella complessità propria dell'intero algoritmo quindi si può sostituire al termine c il termine H pari alla costante scelta come valore prefissato; la complessità si riduce quindi a $O(H^2n^3)$ ove H è fissato e minore di c;

- utilizzare un modo diverso per valutare le *predecessors' and successors' forces*; in particolare i tre tipi di forze sono calcolate in tre fasi successive; nella prima fase vengono calcolate e memorizzate tutte le *self-forces*; nella seconda fase il SDG è attraversato dall'inizio alla fine e per ciascuna operazione la forza memorizzata è posta uguale alla propria *self-force* sommata a quella dei predecessori; infine con una terza passata, questa volta dal basso verso l'alto, vengono sommate anche le forze dei successori; la complessità complessiva scenderebbe secondo quanto riportato in [PK87] a $O(cn^2)$, se si trascura la somma delle forze di predecessori e successori; infatti non trascurando questo aspetto, la complessità si può calcolare come: n iterazioni per n operazioni per c passi di controllo per $\log cn$ somme di forze di predecessori e successori per un totale di $O(cn^2 \log cn)$; applicando la semplificazione del punto precedente si ottiene $O(Hn^2 \log Hn)$ con H costante.

3.2 Estensioni all'algoritmo proposte da Paulin e Knight

Sin dalla presentazione dell'algoritmo, Paulin e Knight illustrarono una serie di estensioni dell'algoritmo (oltre a quelle presentate in 3.1.4 per ridurre la complessità) per introdurre costrutti ciclici, per estendere le finalità dell'algoritmo, per eliminare alcune delle limitazioni della versione base e per tentare di incorporare tecniche di *look-ahead*.

3.2.1 Scheduling con cicli

L'algoritmo descritto fino a questo momento è in grado di gestire tra i costrutti di controllo solo quelli di tipo condizionale. L'estensione che comprende la gestione dei costrutti iterativi (gestione dei cicli) parte dall'assunzione che nella descrizione comportamentale del sistema ad ogni ciclo venga assegnato un vincolo relativo alla durata temporale di una sua iterazione o al numero di unità funzionali allocabili per la sua esecuzione. Nel primo caso il corpo del ciclo

verrà schedulato separatamente utilizzando la versione base dell'algoritmo, nel secondo verrà invece utilizzato il *force-directed list scheduling* (3.2.5).

Nel caso di più cicli annidati, le operazioni relative a quello più interno verranno schedulate per prime, quindi il ciclo stesso ed in particolare una sua iterazione verrà considerata come un'unica operazione avente tempo di esecuzione pari alla durata di un'iterazione. Le operazioni esterne al ciclo non potranno essere schedulate contemporaneamente all'operazione rappresentante il ciclo intero. Questo processo è ripetuto per tutti i cicli finchè viene schedulato quello più esterno.

Per cercare di forzare il pipelining fra le diverse iterazioni del ciclo ed estrarre quindi un maggior parallelismo è possibile invece di schedulare le operazioni della singola iterazione, replicare tali operazioni un numero di volte pari al numero di iterazioni che si vuole vengano eseguite parallelamente. Una volta fatto ciò è sufficiente applicare l'algoritmo classico del Force Directed scheduling sulle operazioni originali e su quelle ottenute tramite la replicazione considerando oltre alle dipendenze originarie quelle presenti fra le operazioni appartenenti a iterazioni diverse.

3.2.2 Estensioni allo scopo dell'algoritmo

Lo scopo primario dell'algoritmo è quello di minimizzare l'utilizzo delle unità funzionali utilizzate; tuttavia è facilmente utilizzabile per minimizzare altri tipi di risorse oltre alle unità funzionali.

3.2.2.1 Minimizzazione dei costi relativi ai bus

Uno degli elementi che è possibile minimizzare utilizzando questo algoritmo sono i costi relativi ai bus all'interno dell'architettura sintetizzata. Schedulare un'operazione in un passo di controllo implica anche assegnare per quel passo di controllo un trasferimento di dati fra l'unità funzionale e un registro (memorizzazione del risultato dell'operazione) e almeno un trasferimento da registro ad unità funzionale (caricamento dei dati dell'operazione). Il numero minimo di interconnessioni necessario per sintetizzare uno scheduling sarà pari al massimo numero di trasferimenti presenti in un passo di controllo. Si può modelliz-

zare l'utilizzo di un'interconnessione come l'utilizzo di un nuovo tipo di unità funzionale. Per calcolare la probabilità di questo tipo di operazioni in un passo di controllo è sufficiente considerare la probabilità di ciascuna operazione di qualsiasi tipo in quel passo di controllo moltiplicata per il numero di interconnessioni distinte per quel tipo di operazione. Quindi per ottenere le somme di probabilità per le interconnessioni è sufficiente applicare la formula:

$$Com\_DG(i) = \sum_{TypeOp} (DG(i, TypeOp) \cdot NofCon(TypeOp)) \tag{3.4}$$

ove

TypeOp è il tipo di operazione;

DG(i, TypeOp) è la somma di probabilità dell'i-esimo passo di controllo dell'-operazione di tipo *TypeOp*;

NofCon(TypeOp) è il numero di connessioni dell'unità funzionale che esegue le operazioni di tipo *TypeOp*.

Nel calcolo delle *self-forces* e delle *predecessors' and successors' forces* bisognerà quindi aggiungere anche le forze relative alle connessioni calcolate allo stesso modo di quelle delle altre operazioni. In questo modo l'algoritmo tenderà a minimizzare sia l'utilizzo di unità funzionali, sia l'utilizzo delle connessioni. Se si vuole minimizzare solo il costo delle connessioni sarà sufficiente considerare solo le forze relative alle interconnessioni e trascurare le altre.

3.2.2.2 Minimizzazione dei costi relativi ai registri

Un secondo parametro che l'algoritmo può minimizzare all'interno dell'architettura è il costo totale dei registri che è pari al loro numero, il quale a sua volta corrisponde al numero massimo di archi di dipendenza dato che attraversano quella linea immaginaria che separa due diversi control step nel SDG schedulato. L'algoritmo applicato a questo problema non solo cerca di minimizzare l'utilizzo dei registri, ma fornisce anche ad ogni iterazione una stima del limite inferiore del numero di registri che sono necessari.

Anche in questa versione dell'algoritmo viene creato un nuovo tipo di operazioni che verrà chiamato *operazione di memorizzazione*: ad ogni operazione che produce un risultato utilizzato da altre operazioni si associa una nuova operazione di tipo memorizzazione indipendentemente dal numero di operazioni che utilizzeranno tale risultato. Tale operazione rappresenta l'esistenza di una variabile in quel determinato passo di controllo. Come costruire la distribuzione di probabilità di questo tipo di operazioni fittizie è tuttavia più complesso di quelle reali o di quelle create nel caso precedente. Nel caso sia l'operazione a monte, sia tutte le operazioni a valle siano state schedulate la distribuzione di probabilità (in questo caso è improprio parlare di distribuzione di probabilità perchè la somma dei diversi valori può essere superiore a uno, ma per affinità si continuerà a chiamarla con questo termine) avrà valore unitario nei passi di controllo compresi fra quello dell'operazione che produce il dato escluso a quello dell'ultima operazione che lo utilizza incluso (cioè si modellizza che i registri, che nella pratica vengono utilizzati nel passaggio fra due passi di controllo, vengano utilizzati nel secondo passo di controllo della coppia a cavallo del passaggio). Negli altri passi di controllo la distribuzione di probabilità varrà 0. Se una o più delle operazioni che individuano il tempo di vita della variabile associata all'operazione di memorizzazione non è schedulata, la creazione della distribuzione di probabilità risulta più complessa, perchè a priori non si conosce quale sarà l'ultima operazione ad utilizzare il dato. Si stima quindi la vita media del dato utilizzando questa formula:

$$TempodiVitaMedio = \frac{Asap + Alap + max}{3} \tag{3.5}$$

dove

- $ASAP$ è il tempo di vita della variabile nello scheduling ASAP;
- $ALAP$ è il tempo di vita della variabile nello scheduling ALAP;
- max è il tempo di vita massimo calcolabile combinando $ASAP_{begin}$(inizio del tempo di vita nell'ASAP) e $ALAP_{end}$(fine del tempo di vita nell'ALAP) come $max = ALAP_{end} - ASAP_{begin} + 1$.

A questo punto i dati raccolti possono aver portato a due diverse situazioni:

1. i tempi di vita forniti da ALAP e ASAP sono disgiunti cioè $ASAP_{end} < ALAP_{begin}$;
2. i tempi di vita forniti da ALAP e ASAP si sovrappongono almeno parzialmente.

Nel primo caso la distribuzione di probabilità dell'operazione di assegnamento varrà $\frac{TempodiVitaMedio}{max}$ nei passi di controllo compresi fra $ASAP_{begin}$ e $ALAP_{end}$, estremi inclusi e 0 negli altri passi di controllo. Nel secondo caso il sovrapporsi di passi di controllo in cui sia per l'ASAP che per l'ALAP una certa variabile sarà viva è indice del fatto che sicuramente con qualsiasi scheduling in quel passo di controllo la variabile sarà viva e quindi in quel passo la distribuzione di probabilità dovrà valere uno. Inoltre questi dati forniscono un'ulteriore informazione: per quel passo di controllo sicuramente quella variabile sarà viva, quindi sarà necessario un registro per memorizzarla ed in questo modo si è ottenuto un'informazione riguardo il numero minimo di registri necessari, anche se magari nessuna operazione è stata ancora schedulata. Negli altri passi di controllo compresi fra $ASAP_{begin}$ e $ALAP_{end}$ ma non facenti parte della sovrapposizione, la distribuzione di probabilità varrà $\frac{TempodiVitaMedio - lunghezzadellasovrapposizione}{max - lunghezzadellasovrapposizione}$. Dalle distribuzioni di probabilità si ricavano poi le somme di distribuzioni di probabilità; a questo punto nel calcolo delle singole forze oltre al contributo delle *self-forces* e delle *predecessors' and successors' forces* delle operazioni canoniche, bisognerà tenere in considerazioni il contributo delle *predecessors' and successors' forces* relative alle operazioni fittizie di memorizzazione utilizzando la formula canonica.

3.2.3 Integrazione di informazioni relative all'architettura

E' possibile sfruttare alcune informazioni relative all'architettura target per indirizzare l'algoritmo di scheduling. Il Force Directed è un algoritmo il cui scopo è minimizzare il numero di risorse utilizzate. In generale l'algoritmo non discrimina un tipo di unità funzionale rispetto ad un altro, ma tende a minimizzare in maniera uniforme il numero di operazioni dei diversi tipi. Tuttavia il costo

di allocazione delle unità funzionali può variare sensibilmente a seconda delle funzionalità implementate. Quindi può essere opportuno cercare di favorire la minimizzazione delle risorse più costose a scapito di quelle economiche. Un modo semplice ed efficace per indirizzare l'algoritmo di scheduling in questa direzione rendendo più critiche le operazioni più costose è moltiplicare le somme di probabilità per un fattore indice del costo del particolare tipo di unità. In questo modo l'algoritmo sarà portato a cercare di schedulare prima le operazioni relative a unità funzionali con costo maggiore e quindi minimizzare con maggiore efficacia l'occupazione delle risorse più costose.

Una seconda informazione che è non solo possibile ma fortemente consigliato utilizzare all'interno del Force Directed per ottenere risultati migliori è quella relativa alla presenza di unità funzionali che possono compiere diversi tipi di operazione come ad esempio le ALU. Per inserire questa informazione all'interno dei dati dell'algoritmo è sufficiente utilizzare in ogni passo di controllo al posto di una somma di probabilità per ciascun tipo di operazione eseguibile dall'unità multifunzionale un'unica somma che tenga conto delle probabilità di tutte le operazioni assegnate a quel tipo di risorsa. Per ottenere questo valore non è sufficiente sommare le somme di probabilità relative ai singoli tipi di operazione assegnati a quell'unità funzionale: in questo modo infatti non si terrebbe conto della possibilità che esistano mutue esclusioni fra le operazioni appartenenti alle diverse somme che portano ad un calcolo differente della somma di probabilità come illustrato in 3.1.1; il metodo corretto per calcolare la somma di probabilità relativa a unità multifunzonali è invece sostituire tutte le operazioni assegnate a quel tipo di risorsa con un unico nuovo tipo fittizio, che per esempio potrebbe essere chiamato con il nome stesso del tipo di unità, e proseguire nell'applicazione dell'algoritmo con i soliti passi. Questa semplificazione si può attuare però a condizione che:

- nel caso si stia minimizzando il costo dei bus, le operazioni afferenti alla stesso tipo di unità funzionale abbiano un numero uguale di ingressi;

- le operazioni abbiano tempo di esecuzione e tempo di set-up relativo (confrontare quanto verrà esposto in 3.2.4) uguale tra loro.

In caso contrario non è possibile applicare la sostituzione e sarà necessario calcolare la somma di probabilità dell'unità funzionale multipla utilizzando le operazioni originarie e tenendone quindi in conto le caratteristiche.

3.2.4 Rilassamento di alcune condizioni per l'applicazione dell'algoritmo

Nella prima descrizione dell'algoritmo (3.1) si è posto come limitazione che tutte le operazioni venissero eseguite in un unico ed intero passo di controllo. Questa restrizione è tuttavia superabile ed in particolare l'algoritmo del Force Directed può considerare l'esistenza di operazioni concatenate (coppia o numero maggiore di operazioni eseguibili in sequenza all'interno del periodo di un passo di controllo e legate da dipendenze di dato), operazioni multiciclo (operazioni che necessitano di più passi di controllo per essere eseguite e che durante tutti i passi di controllo rendono indisponibile la risorsa) e operazioni eseguite da unità funzionali pipeline.

3.2.4.1 Scheduling con Chaining

Per permettere scheduling con chaining verrà aumentata la mobilità calcolata con ASAP e ALAP: l'ASAP verrà anticipato di uno o più passi di controllo se è possibile concatenare l'operazione in oggetto con una o più delle operazioni precedenti e simmetricamente l'ALAP verrà posticipato di uno o di più passi di controllo se è possibile concatenare l'operazione in oggetto con una o più successive. Per determinare se ci sono delle operazioni concatenabili si associa ad ogni operazione la sua latenza, quindi si esegue l'ASAP memorizzando per ogni operazione la somma delle latenze del percorso critico precedente quell'operazione (in pratica all'ASAP basato sui passi di controllo si affianca l'ASAP basato sui tempi di esecuzione). Confrontando l'ASAP di un'operazione basato sui tempi di esecuzione sommato al tempo di esecuzione proprio dell'operazione (per così dire l'ASAP del termine e non dell'inizio dell'operazione) con gli ASAP temporali dei predecessori, se la differenza di questi valore è inferiore alla durata del priodo di un passo di controllo allora questa coppia di oper-

azioni, e quelle eventualmente comprese fra di esse nel SDG, possono essere concatenate cioè schedulate nello stesso passo di controllo.

3.2.4.2 Scheduling di operazioni multiciclo

Così come il Force Directed prevede la possibilità del concatenamento delle operazioni, esso prevede anche la possibilità di schedulare operazioni che richiedano più di un passo di controllo per essere eseguite (operazioni multiciclo). Le modifiche da applicare all'algoritmo sono:

- Calcolo della mobilità

  Le uniche differenze che si deve tenere in conto nel calcolo della mobilità sono quelle previste negli algoritmi ASAP e ALAP considerando anche operazioni con latenza superiore al singolo passo di controllo; tuttavia è possibile combinare questa caratteristica con quella della concatenazione delle operazioni; le catene di operazioni concatenate in questo caso potranno occupare più di un passo di controllo, ma per semplificare le strutture di controllo e il calcolo dell'utilizzo delle unità funzionali si considera solo il caso in cui per schedulare la catena di operazioni sia necessario un numero di passi di controllo comunque pari a quello necessario a schedulare l'operazione più lunga della catena; cioè data un'operazione con tempo di esecuzione superiore a quello del ciclo di controllo, ma che non è multiplo della durata del ciclo stesso, si cercherà di concatenare ad essa una o più operazioni per eventualmente sfruttare il tempo rimanente a disposizione nel primo o nell'ultimo o in entrambi i passi di controllo dello scheduling dell'operazione lunga; l'utilizzo della concatenazione deve comunque essere considerato come una possibilità da valutare e non come un'imposizione in quanto scopo dell'algoritmo è la minimizzazione delle risorse e non del tempo totale di esecuzione e la concatenazione non necessariamente riduce il numero di risorse necessarie.

- Calcolo delle distribuzioni di probabilità

  Così come nel caso dei registri, anche la distribuzione di probabilità di operazioni multiciclo perde la proprietà matematica di avere somma uni-

taria. La somma delle distribuzioni di probabilità di un'operazione multiciclo sarà infatti pari al numero stesso di cicli necessari per eseguire l'operazione. Un'operazione con mobilità unitaria e tempo di esecuzione pari a n passi di controllo avrà distribuzione di probabilità così impostata: 1 nell'unico passo di controllo in cui è assegnabile l'operazione, ma 1 anche negli n-1 passi successivi. Per operazioni aventi mobilità non nulla il calcolo è più complesso: in pratica è come se le si scomponessero in un numero di frazioni pari al numero di passi di controllo necessari ad eseguirle e si creasse una distribuzione di probabilità per un'operazione nella sua completezza non direttamente ma costruendo prima una distribuzione di probabilità per ciascuna delle frazioni create. Una volta fatto ciò la distribuzione di probabilità relativa all'operazione complessiva si ottiene sommando per ogni passo di controllo i contributi relativi alle singole frazioni dell'operazione. La distribuzione di probabilità della prima frazione è quella che si avrebbe nel caso di un'operazione normale cioè un valore pari a $\frac{1}{mobilita'}$ per ogni passo di controllo tra l'ASAP e l'ALAP della operazione estremi compresi. Le successive frazioni dell'operazione devono essere schedulate nei passi successivi consecutivi a quello della prima frazione, quindi le corrispondenti distribuzioni di probabilità saranno uguali a quelli della prima frazione ma traslate opportunamente in avanti di un numero di passi di controllo pari all'indice della frazione meno uno.

Una volta applicate queste modifiche è possibile applicare normalmente l'algoritmo.

3.2.4.3 Scheduling su unità pipelined

L'introduzione della possibilità di utilizzare unità funzionali dotate di pipeline è molto semplice. Si considera che le operazioni eseguite da esse occupino la risorsa e quindi vengano logicamente schedulate solo nel primo ciclo della loro esecuzione. Dal punto di vista dell'algoritmo quindi un'operazione di questo tipo verrà trattata esattamente come un'operazione che possa venir eseguita in un unico passo di controllo. Un'altra interpretazione possibile è quella di considerarla come un'operazione multiciclo ma porre uguale a zero la probabilità

che le frazioni successive alla prima vengano schedulate in qualsiasi passo di controllo. Chi dovrà fare distinzione di questo tipo di operazioni e tenere conto del ritardo fra il momento in cui un'operazione viene schedulata e il momento in cui si rende disponibile il suo risultato alle operazioni successive sono gli algoritmi ASAP e ALAP nel corso del calcolo delle finestre temporali e della mobilità.

3.2.5 Estensione con vincoli sulle risorse

3.2.5.1 Il *force-directed list scheduling*

Paulin e Knight hanno proposto un metodo per utilizzare il meccanismo del calcolo delle forze per realizzare uno scheduling con vincoli sulle risorse che punti a minimizzare i passi di controllo ovvero un algoritmo duale di quello fin qui illustrato. L'algoritmo illustrato prende il nome di *Force-Directed List Scheduling* in quanto è una composizione dell'algoritmo Force-Directed e dell'algoritmo *List-Based* (2.2.1.4).

L'approccio del *force-directed list scheduling* è simile a quello del *list-based* tradizionale: la differenza sostanziale consiste nella funzione di priorità utilizzata; nel caso in un passo di controllo il numero di operazioni di un certo tipo pronte e quindi schedulabili sia superiore al numero di risorse disponibili è necessario scegliere di quali operazioni posticipare lo scheduling; la funzione di priorità utilizzata è la forza relativa al restringimento della finestra temporale dell'operazione da quella attuale ad una che escluda il passo di controllo corrente; le operazioni con forza minore sono quelle che se ritardate non aumenterebbero la congestione nell'utilizzo delle unità funzionali nei passi successivi a quello corrente o lo farebbero in misura minore. Una volta calcolato quale operazione ha forza minore, essa viene posticipata; se vi fossero ancora troppe operazioni pronte ad essere schedulate non si sceglierà l'operazione con forza minore già calcolata fra quelle sopravvissute, ma sarà necessario prima di effettuare una nuova scelta ricalcolare le forze interessate e ripetere questo procedimento finchè abbastanza operazioni siano state posticipate. Il ricalcolo è reso non superfluo dal fatto che modificando la mobilità di un'operazione vengono modificate anche le mobilità dei suoi successori, cosa che può modificare il

valore delle forze delle operazioni pronte. Il posticipare le operazioni non implica automaticamente che esse assumano priorità maggiore rispetto a quelle passibili di scheduling a partire dal passo di controllo successivo e che esse vengano quindi schedulate nel passo di controllo successivo. Nel caso infatti nel passo successivo non vi fossero nuovamente abbastanza unità funzionali disponibili per soddisfare tutte le operazioni pronte si dovrà ricorrere nuovamente al calcolo delle forze per le operazioni in conflitto. Rimane da illustrare come calcolare le forze. Infatti a differenza dell'algoritmo presentato in 3.1 (l'algoritmo nella sua versione base) non si hanno informazioni riguardo alla mobilità delle operazioni, ne riguardo al numero totale di passi di controllo necessari. Paulin e Knight hanno deciso di utilizzare per questo valore la lunghezza in passi di controllo del percorso critico corrente (esso può infatti aumentare se una delle operazioni critiche è stata posticipata). Stabilito questo valore si applicano gli algoritmi ASAP e ALAP (utilizzando sempre come numero di passi di controllo quello relativo al percorso critico) e quindi si ricavano finestre temporali e mobilità. Evidenziate queste particolarità si può riassumere i passi principali dell'algoritmo:

1. porre il numero di passi controllo pari alla lunghezza del percorso critico misurato in numeri di cicli di controllo;

2. per ogni passo di controllo:

   (a) calcolare con ASAP e ALAP le finestre di tempo delle operazioni;

   (b) determinare le operazioni che sono pronte nel passo di controllo corrente (operazioni i cui predecessori sono già state schedulati nei passi di controllo precedenti);

   (c) finchè il numero di operazioni pronte di un certo tipo supera il numero di unità funzionali che possono eseguire quel particolare tipo di operazione;

      i. se tutte le operazioni appartengono al percorso critico il numero totale di passi di controllo deve essere incrementato di un'unità e quindi tutte le finestre temporali dovranno essere ricalcolate;

ii. calcolare le forze dovute al posporre le operazioni del tipo in oggetto;

iii. posticipare lo scheduling dell'operazione con minor forza;

iv. rimuovere l'operazione dalla lista di quelle pronte;

(d) schedulare nel passo di controllo corrente tutte le operazioni pronte.

Questo approccio combina l'alto utilizzo delle unità funzionali, la bassa complessità computazionale ($O(n^2)$ nel caso pessimo, $O(n)$ mediamente) e la valutazione globale di tutti gli effetti collaterali di assegnare un'operazione ad un passo di controllo.

Un modo per sfruttare appieno questa versione dell'algoritmo è quella di utilizzarlo in combinazione con la formulazione originale: in una prima fase si fissa una stima del numero di passi di controllo necessari allo scheduling se esso non è già stato fissato come requisito dell'implementazione che si vuole ottenere; utilizzando questo dato si applica l'algoritmo Force Directed, in particolare integrandovi informazioni relative all'architettura come mostrato in 3.2.3, per ottenere una stima del numero minimo di unità funzionali necessarie per schedulare tutte le operazioni. A questo punto si applica il force-directed list scheduling utilizzando come numero di risorse quello appena stimato per verificare se non fosse possibile ottenere uno scheduling in un numero di passi di controllo inferiore. Questa combinazione dei due algoritmi permette quindi di esplorare lo spazio di progetto, seppure in due passate, cercando di minimizzare due metriche diverse, tempo e area, che sono solitamente in contrapposizione.

3.2.5.2 Calcolo del Numero di Passi di Controllo per Tentativi

Esiste un'altra proposta di Paulin e Knight per utilizzare il Force Directed scheduling con vincoli sulle risorse per minimizzare il numero di passi di controllo. La struttura di questa versione dell'algoritmo è la seguente: si individua con ASAP e ALAP il numero minimo di passi di controllo necessario allo scheduling e si applica l'algoritmo classico su questi dati. Se durante la sua esecuzione si verifica che la stima (non il valore relativo alle operazioni già

schedulate) di utilizzo di un tipo di unità funzionale superasse il numero di unità effettivamente disponibili, si incrementa di un'unità il numero di passi di controllo totale e conseguentemente si incrementa di uno anche la mobilità di tutte le operazioni estendendone la finestra temporale. Ciò viene applicato non solo alle operazioni ancora da schedulare, ma anche a quelle già schedulate i cui assegnamenti vengono così eliminati. Le finestre temporali risultano comunque più piccole di quelle che si otterrebbero applicando inizialmente ASAP e ALAP con numero di passi di controllo pari a quello attuale. In questo modo si ha un aumento del tempo di esecuzione dell'algoritmo, ma tendenzialmente non significativo perchè la sua complessità rimane inalterata in quanto le finestre temporali non vengono ricalcolate, ma semplicemente estese.

3.2.6 Introduzione di vincoli temporali locali

In alcune classi di applicazioni esistono dei vincoli temporali fra coppie di operazioni cioè vincoli sul tempo minimo e massimo che debba intercorrere fra le due computazioni. Questa richiesta è modellizzabile all'interno del Force Directed inserendo delle finte operazioni temporizzanti nel SDG fra le due operazioni incriminate ed associando a queste operazioni fittizie un intervallo di tempo. Esse in un certo senso servono solo come prememoria: non verranno mai schedulate, ma verranno tenute in considerazione per il calcolo delle finestre temporale delle operazioni che collegano, in modo tale che tali finestre rispettino i vincoli memorizzati nell'operazione temporizzante.

3.2.7 Tecniche di look-ahead

Per le caratteristiche dell'algoritmo Force Directed, il costo in termini di complessità temporale di utilizzare tecniche di look-ahead non è compensato dai minimi guadagni ottenibili con esse. Tuttavia gli autori stessi hanno proposto un'approssimazione di una tecnica di look-ahead (confrontare 2.3.1) che non comporta un aggravio della complessità dell'algoritmo, ma che ne migliora, a dire degli autori, considerevolmente l'efficienza. L'idea in questo caso è di utilizzare all'interno della formula per il calcolo delle *self-forces* non il valore attuale

della somma di probabilità, ma una media pesata fra quello attuale e quello che assumerà dopo l'assegnamento; nel dettaglio la formula utilizzata è:

$$Force(i) = DG\_temp(i) \cdot x(i)$$

dove

Force(i) è il contributo alla forza dello scheduling relativo all'i-esimo passo di controllo;

DG_temp(i) è la somma di probabilità del tipo di unità funzionale che può eseguire l'operazione nell'i-esimo passo di controllo modificata per tenere in considerazione la situazione futura in questo modo: $DG\_temp(i) = DG(i) + x(i)/3$;

x(i) è la variazione della probabilità dell'operazione nell'i-esimo passo di controllo a seguito dello scheduling.

Tuttavia svolgendo la formula si ottiene

$$Force(i) = (DG(i) \cdot x(i)) + ((x(i))^2)/3$$

Il primo termine è presente già nella prima formulazione, quindi soffermandosi sul secondo si può notare che la differenza fra la forza calcolata nel modo tradizionale e quella calcolata con quest'ultimo metodo è pari a

$$\triangle\ SelfForce = \sum_i \frac{x^2(i)}{3} \tag{3.6}$$

Analizzando questa formula è immediato constatare che essa non dipenda dalle somme di probabilità ma solamente dalle variazioni nelle distribuzioni di probabilità dell'operazione nei vari passi. Queste variazioni non dipendono dagli specifici passi di controllo in cui è possibile schedulare l'operazione, ne dallo specifico passo in cui si sceglie di assegnare l'operazione o dal suo tipo ma unicamente dalla mobilità posseduta da essa. La formula può essere quindi riscritta in questo modo, indicando con m la mobilità dell'operazione prima

dello scheduling:

$$\triangle\ SelfForce = \frac{(\frac{m-1}{m})^2 - (m-1)(\frac{1}{m})^2}{3} = \frac{(m^2 - 3m + 2)}{3m^2} \tag{3.7}$$

Quello che si è ottenuto è quindi una correzione delle *self-force* che dipende unicamente dalla mobilità dell'operazione in esame e che quindi non influenza direttamente la scelta del passo di controllo di una certa operazione, ma tuttalpiù l'ordine in cui viene scelta la prossima operazione da assegnare.

Capitolo 4

Il Force Directed Scheduling proposto

In questo capitolo verranno illustrate le modifiche apportate all'algoritmo originale del *Force Directed scheduling*. Nella prima sezione verranno mostrate le variazioni applicate ai passi dell'algoritmo originario basate sulle osservazioni fatte nel capitolo precedente per ottenere migliori risultati in termini di numero di risorse allocate o di tempi di esecuzione minori. Nella seconda sezione verrà mostrata la versione proposta dell'algoritmo che consente di considerare contemporaneamente vincoli temporali e vincoli tecnologici a differenza delle estensioni proposte dagli autori che prevedevano la presenza di vincoli di un unico tipo.

4.1 Proposte di modifiche all'algoritmo originale

Le variazioni che verranno di seguito descritte sono state ipotizzate a seguito dell'analisi e delle critiche fatte alla versione originale dell'algoritmo di cui sono stati evidenziati (in particolare in 3.1.2, 3.1.3 e 3.2.7)) i limiti insiti nel gestire particolari soluzioni.

4.1.1 Priorità delle operazioni

Il *Force Directed scheduling* è un algoritmo che presenta una complessità computazionale relativamente elevata rispetto ad algoritmi più semplici quali l'ASAP, l'ALAP o il List Based. Questa maggiore complessità è da ascriversi principalmente alla visione più globale che questo algoritmo ha rispetto a quelli citati. Esso infatti per decidere un assegnamento deve tenere conto di tutti gli effetti secondari che tale assegnamento avrebbe nei diversi passi di controllo. Inoltre non segue un ordine prefissato nel scegliere quale operazione schedulare e quindi necessita che ad ogni iterazione le informazioni relative a tutte le operazioni vengano ricalcolate. E' evidente se si confronta questo sistema con quello per esempio del List Based in cui vengono considerate ad ogni passo di controllo solo le operazioni considerate pronte quanto questo comporti un dilatarsi dei tempi di calcolo.

Gli autori stessi dell'algoritmo (come mostrato in 3.1.4 - considerazioni sulla complessità) hanno cercato soluzioni che ne riducessero la complessità limitando le finestre temporali delle operazioni. La soluzione qui presentata prevede invece una riduzione del numero di candidate ad essere schedulate in ogni iterazione, in modo tale che l'algoritmo dovendo scegliere solo da un sottoinsieme di operazioni, debba calcolare le forze solo relativamente a questo sottoinsieme. Terminato di assegnare le operazioni del sottogruppo prescelto si selezionerà un nuovo sottogruppo e così via sino ad esaurire tutte i sottogruppi e quindi tutte le operazioni. Questa limitazione applicata alla versione completa dell'algoritmo può essere anche combinata con quella proposta in 3.1.4 per ottenere un'ulteriore riduzione della complessità a spesa della bontà dei risultati. Il problema di ridurre il numero delle candidate raggruppandole in sottogruppi può essere modellizzato assegnando ad ogni operazione una priorità: ad ogni iterazione verranno calcolate solo le forze delle operazioni con priorità maggiore che saranno le possibili candidate ad un assegnamento. Rimane da determinare come scegliere le priorità cioè quali operazioni raggruppare in modo tale che vengano schedulate in successione. Per far questo si può considerare che alla base del Force Directed scheduling c'è il calcolo degli effetti che ha un assegnamento di un'operazione su tutte le altre operazioni in tutti i passi

di controllo.

Tuttavia ci si può chiedere se effettivamente un assegnamento abbia un'influenza significativa così estesa. Nella realtà si può facilmente evincere che se due operazioni sono molto distanti nel SDG l'assegnamento di una non influenzerà i possibili assegnamenti dell'altra o lo farà in misura minore. Una prima indicazione su come raggruppare l'operazioni è quindi quella data dalla loro vicinanza all'interno del SDG: operazioni vicine avranno finestre temporali sovrapposte e quindi concorreranno per l'utilizzo delle unità funzionali negli stessi passi di controllo. Questo purchè le operazioni non siano in mutua esclusione reciproca, cioè non appartengano per esempio al ramo then e al ramo else di un costrutto condizionale. Infatti nel caso le operazioni siano in mutua esclusione, esse non saranno in concorrenza per l'assegnamento di unità funzionali quindi si può pensare di schedularle in momenti diversi e perciò assegnarli a gruppi diversi.

Infine è possibile fare una considerazione sulle operazioni di salto condizionale: esse costituiscono per così dire dei paletti che fissano lo scheduling di tutte le altre operazioni, sia perchè tipicamente hanno bassa mobilità facendo parte dei percorsi critici o dei rami ad essi più vicini, sia perchè molte operazioni hanno una dipendenza di controllo da esse.

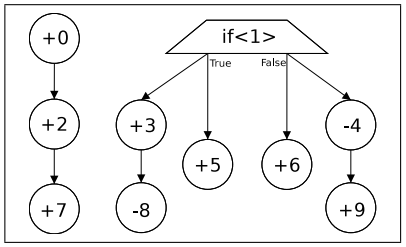
Sulla base di questa serie di considerazioni si è scelto di utilizzare come suddivisione delle operazioni quella fornita dai blocchi basici che ovviamente rispetta sia il fatto di raggruppare operazioni vicine nei SDG, sia il fatto di raggruppare operazioni non in mutua esclusione reciproca. Questa suddivisione ha tuttavia la limitazione di essere di fatto nulla nel caso di assenza di costrutti condizionali. In questo caso l'algoritmo non si discosta dalla versione originale. In 4.2.5 (priorità delle operazioni in presenza di vincoli) verrà mostrato un possibile ordinamento parziale dei blocchi basici e quindi dei sottogruppi, ma tale ordinamento ha il solo scopo di diminuire i tempi di esecuzione in presenza di vincoli tecnologici: allo stato attuale non si ha indicazione se tale ordinamento offra risultati migliori, peggiori o complessivamente non confrontabili di un ordinamento completamente casuale. Individuare il migliore ordinamento dei blocchi basici è infatti a tutt'oggi ancora un problema insoluto. In mancanza di certezze viene ora utilizzato l'ordinamento parziale mostrato in 4.2.5 in quanto

è plausibile ritenere che questo ordinamento provocando un minore numero di violazioni di vincoli in quel caso per le ragioni ivi riportate, produca nel caso privo di vincoli risultati tendenzialmente migliori; come verrà illustrato l'ordinamento è parziale. Per renderlo totale si utilizzerà come seconda chiave della funzione di ordinamento (la prima è appunto la posizione nell'ordinamento parziale) una funzione che ordini in modo fissato ma casuale i blocchi basici.

4.1.2 Interpretazione sulla formula della forza

Consideriamo l'esempio riportato in figura 4.1: in 4.1(a) è riportato un SDG di esempio con un'operazione di tipo salto condizionale; in 4.1(b) sono riportate le distribuzioni di probabilità (l'etichette T ed F indicando se le operazioni fanno parte del ramo *then* o del ramo *else* del costrutto condizionale) e le somme di probabilità. Applicando l'algoritmo otteniamo una forza nulla per i quattro possibili assegnamenti: quale di questi si deve scegliere? Si potrebbe pensare che la scelta essendo le forze tutte uguali sia indifferente. Provando a compiere due scelte diverse, quindi a calcolare le forze per l'operazione rimanente e a schedularla si ottengono i risultati mostrati in 4.1(d) e 4.1(e). Se valutiamo i due scheduling, trascurando le unità funzionali necessarie per sottrazione e costrutto condizionale che sono in numero uguale in entrambi i casi, si può notare come nel primo caso siano necessari tre unità funzionali in grado di eseguire addizioni, nel secondo caso solo due. A questo punto si potrebbe pensare che per compiere la scelta migliore basterebbe, nel caso si presentino più forze eguali, considerare altri fattori come l'effettiva occupazione dopo l'assegnazione o introdurre tecniche di look-ahead.

A questo punto il problema di scheduling in oggetto viene modificato aggiungendo nuove operazioni come mostrato in 4.2(a). Applicando in questo caso l'algoritmo non si hanno casi di indecisione perchè la scelta che propone di fare l'algoritmo (4.2(c)) è chiaramente di assegnare l'operazione *+6* al passo di controllo 2. Il risultato di questa decisione è mostrato in 4.2(d): tale soluzione utilizza un sottrattore, un moltiplicatore e tre addizionatori. Tuttavia eseguendo lo scheduling manualmente si individua facilmente la soluzione riportata in 4.2(e) che utilizza un sommatore in meno. Alla luce di questo esempio anche la



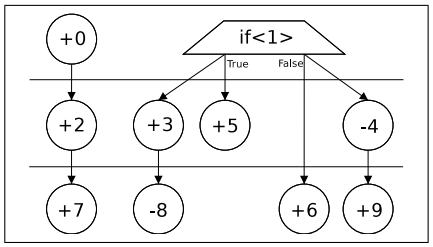
(a) SFG di esempio

Step	Distribuzione di Proabilità delle Operazioni	Somma
0	+ if	+: 1.0 if: 1.0 -: 0.0
1	+ +T + T + F -F	+: 2.5 if: 0.0 -: 1.0
2	+ -T +F	+: 2.5 if: 0.0 -: 1.0

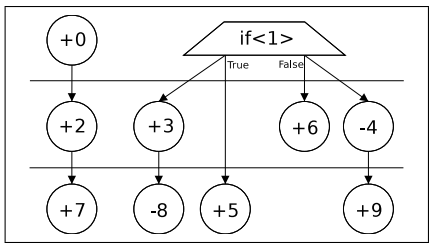
(b) Distribuzione di probabilità e somma di probabilità

Op	CS	SF	OF	TF
+5	1	0.0	0.0	0.0
+5	2	0.0	0.0	0.0
+6	1	0.0	0.0	0.0
+6	2	0.0	0.0	0.0

(c) Tabella delle forze delle operazioni da schedulare
SF = self forces
OF = predecessors' and successors' forces
TF = total forces

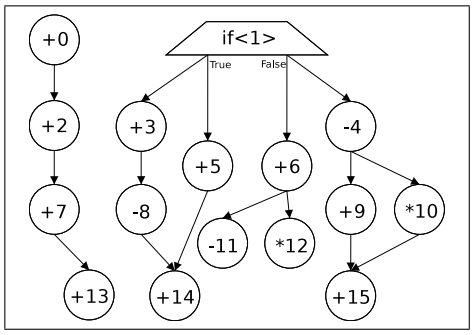


(d) Esempio di possibile scheduling prodotto dal Force Directed - caso pessimo

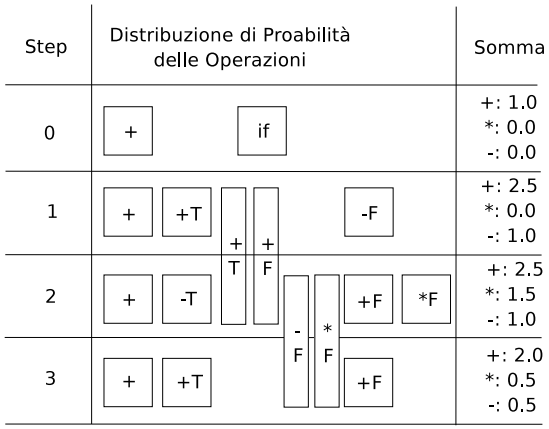


(e) Esempio di possibile scheduling prodotto dal Force Directed - caso ottimo

Figura 4.1. Esempio di indecisione nella scelta di un assegnamento in caso di costrutti condizionali



(a) SDG di esempio

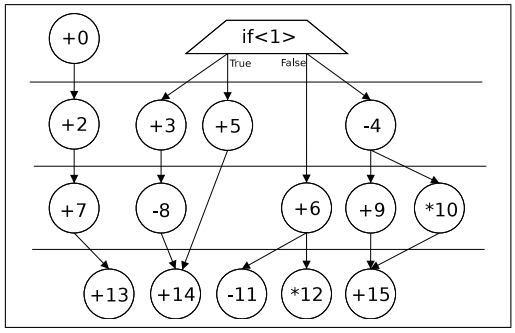


(b) Distribuzione di probabilità e somma di probabilità

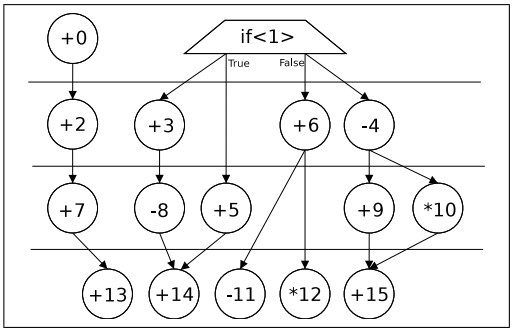
Op	CS	SF	OF	TF
+5	1	0.0	0.0	0.0
+5	2	0.0	0.0	0.0
+6	1	0.0	0.0	0.0
+6	2	0.0	-0.75	-0.75

Op	CS	SF	OF	TF
-11	2	+0.25	0.0	+0.25
-11	3	-0.25	0.0	-0.25
*12	2	+0.5	0.0	+0.5
*12	3	-0.5	0.0	-0.5

(c) Tabella delle forze delle operazioni da schedulare
SF = self forces
OF = predecessors' and successors' forces
TF = total forces



(d) Scheduling prodotto dal Force Directed



(e) Scheduling ottimo

Figura 4.2. Esempio di scelta di assegnamento svantaggiosa in caso di costrutti condizionali

possibilità di introdurre un raffinamento dell'algoritmo in caso di indecisione appare inutile perchè possono presentarsi situazioni simili in cui la scelta suggerita è unica, ma palesemente svantaggiosa. Introdurre una tecnica di look-ahead completa che venisse utilizzata in tutte le occasioni è stato dimostrato essere troppo penalizzante in 3.2.7 (tecniche di look-ahead). Conviene quindi analizzare questo particolare tipo di situazioni e chiedersi perchè l'algoritmo suggerisca scelte penalizzanti.

L'algoritmo tiene conto della presenza di istruzioni in mutua esclusione solo nel calcolo delle somme di probabilità: quando si tratta di calcolare la forza di un'operazione tuttavia non si considera come è stata calcolata la somma di probabilità. Quindi il contributo alla forza di un'operazione relativo ad un passo di controllo tiene conto solo della somma di probabilità stessa e della mobilità dell'operazione. Questo comporta che l'algoritmo in realtà per calcolare la forza (gli effetti di un assegnamento di un'operazione a un dato passo di controllo non tenga conto del fatto che l'operazione assegnata sia effettivamente in conflitto con quelle già presenti o meno. Schedulare in un passo di controllo un'operazione appartenente ad un ramo then di un costrutto di controllo se già sono presenti in quel passo di controllo più operazioni dello stesso tipo appartenenti al ramo then stesso e nessuna al ramo else porta ad effetti (è richiesta un'ulteriore unità funzionale di quel tipo per quel passo di controllo) differenti rispetto a schedulare nella stessa situazione un'operazione appartenente al ramo else (non sono richieste ulteriori unità funzionali per quel passo). Questa sostanziale differenza non è tenuta in alcun conto dalla versione originale dell'algoritmo che cerca quindi di non schedulare contemporaneamente operazioni del ramo then e del ramo else di un costrutto condizionale anche se esse non concorrono ovviamente per l'accesso ad una risorsa essendo in mutua esclusione.

Per ovviare a questa mancanza si è deciso di modificare l'algoritmo originale in un modo che può essere definito sostanziale: in particolare si modifica la formula della forza che ne costituisce il fondamento. La formula originaria è stata infatti modificata sostituendo un termine in questo modo:

$$Force(i) = DG(i) \cdot \overline{x}(i) \tag{4.1}$$

dove

Force(i) è la forza dello scheduling relativa all'i-esimo passo di controllo;

DG(i) è la somma di probabilità del tipo di unità funzionale cui è assegnata l'operazione nell'i-esimo passo di controllo;

$\overline{x}$**(i)** è la variazione non della probabilità dell'operazione nell'i-esima operazione a seguito dello scheduling ma la variazione della somma di probabilità dovuta all'assegnamento.

Ovviamente la sostituzione evidenziata dell'ultimo termine della formula viene fatta sia in nel calcolo della *self-force*, sia nel calcolo delle *predecessors' and successors' forces*. Nel caso non vi siano costrutti condizionali o nel caso non possano venir schedulate istruzioni in mutua esclusione nel passo di controllo in oggetto il calcolo delle forze risulta uguale a quello classico perchè la variazione della probabilità dell'operazione coincide con la variazione della somma di probabilità. La stessa cosa accade in presenza di costrutti condizionali se l'operazione oggetto della modifica di mobilità non sia in mutua esclusione con alcuna delle operazioni dello stesso tipo schedulabili in quel passo di controllo. Rimane da evidenziare cosa accade nel caso l'operazione di cui si sta calcolando una forza sia in mutua esclusione con qualcun altra dello stesso tipo schedulabile in quel passo di controllo; è bene introdurre prima tre definizioni relative alla somma di probabilità prima di analizzare ciò che può accadere; ovviamente tutte le definizioni sottintendono che ci si stia riferendo ad un fissato tipo di operazioni:

- somma di probabilità *totale*:

  la somma di probabilità classica evidenziata in 3.1.1;

- somma di probabilità *relativa* (ad un ramo: esso può essere un ramo then o un ramo else):

  la somma di probabilità ottenuta considerando solo le operazioni che appartengono ad un ramo condizionale;

- somma di probabilità *predominante*:

  dato un costrutto condizionale e dati i rami che si dipartono da esso, è definita somma di probabilità predominante il valore maggiore fra le somme di probabilità relative e ramo predominante il ramo cui appartiene tale forza.

Vengono ora evidenziati i tre casi possibili (le somme di probabilità si riferiscono ovviamente al tipo di unità funzionale che può eseguire l'operazione):

- l'operazione fa parte del ramo condizionale predominante:

  la somma di probabilità complessiva sarà quindi pari a quella di quel ramo sommata al contributo della parte non condizionata del SDG; si possono quindi distinguere due ulteriori sottocasi:

  - aumento di probabilità dell'operazione:

    in questo caso l'aumento di probabilità dell'operazione provoca un eguale aumento della somma di probabilità, quindi la forza sarà positiva perchè potenzialmente vi è un aumento della richiesta d'uso dell'unità funzionale e sarà pari a quella calcolata tradizionalmente;

  - diminuzione di probabilità dell'operazione:

    in questo caso la diminuzione di probabilità dell'operazione provoca una diminuzione della somma di probabilità; la diminuzione è pari a quella della probabiltà stessa a meno che essa non faccia diventare un altro ramo quello predominante; in questo caso la diminuzione della somma è pari alla differenza fra le somme di probabilità originarie relative dei due rami: infatti le risorse necessarie per effettuare lo scheduling in un passo di controllo sono determinate da quello che attualmente è il ramo di controllo dominante (sommato alle probabilità delle operazioni che non fanno parte di alcun ramo condizionale);

- l'operazione non fa parte del ramo condizionale predominante:

  - aumento di probabilità dell'operazione:

    se l'aumento di probabilità non trasforma il ramo non dominante in ramo dominante, la variazione della somma di probabilità è nulla: infatti le risorse allocate per schedulare il ramo dominante basteranno per schedulare il ramo non dominante compreso l'aumento attuale; se l'aumento trasforma il ramo in dominante allora servirà allocare delle risorse suppletive per colmare il gap fra vecchio e nuovo ramo dominante; tale differenza corrisponderà all'aumento della somma di probabilità e sarà pari alla differenza fra le somme di probabilità dei due rami dopo l'aumento;

  - diminuzione di probabilità dell'operazione:

    la diminuzione di probabilità, intervenendo sul ramo non dominante, non varia le risorse da allocare per effettuare lo scheduling, quindi non varia neppure la somma di probabilità;

- i rami hanno la stessa somma di probabilità relativa:

  - aumento di probabilità dell'operazione:

    il ramo a cui appartiene l'operazione diventa automaticamente quello dominante, quindi la somma di probabilità totale deve essere incrementata dell'aumento di probabilità dell'operazione;

  - diminuzione di probabilità dell'operazione:

    il ramo a cui non appartiene l'operazione diventa automaticamente quello dominante, quindi l'operazione apparterà al ramo non dominante e pertanto la somma di probabilità totale rimane inalterata.

Le considerazioni fatte riguardano il caso con un unico costrutto condizionale. Nel caso di più strutture condizionali annidate, il ragionamento è simile e va eseguito iterativamente a partire dai costrutto più interno.

4.1.3 Correzione nel calcolo di *predecessors' and successors' forces*

Come mostrato in 3.1.2 (calcolo delle forze) e come verrà ripreso ed esteso in 4.1.4 vi sono delle approssimazioni nel calcolo delle *predecessors' and successors' forces*. Modificare l'algoritmo in modo tale che esso tenga conto esattamente degli effetti provocati da un possibile assegnamento risulterebbe controproducente per l'elevata complessità che esso assumerebbe che non giustificherebbe l'eventuale guadagno nella qualità dei risultati. L'approssimazione evidenziata produce in generale una sottostima del valore del *predecessors' and predecessors' forces*. Nel caso di forze dal valore positivo l'approssimazione nel calcolo di questa componente della forza non risulta essere particolarmente penalizzante ai fini delle scelte di assegnamento, in quanto la forza totale sarà tendenzialmente comunque positiva e perciò l'assegnamento corrispondente difficilmente sarà selezionato. Nel caso di forze dal valore negativo invece l'approsimazione nel calcolo può portare come mostrato nella figura 3.2 a delle scelte palesemente svantaggiose. Infatti l'approssimazione può trasformare in negative forze in realtà positive e pertanto portare a degli assegnamenti che nel caso di un calcolo corretto non verrebbero effettuati.

Per ovviare a questo problema si è scelto di diminuire il peso dei contribuiti alle *predecessors' and successors' forces* qualora essi fossero negativi in modo tale da renderli trascurabili rispetto a quelli positivi. In questo modo non solo si risolve il problema evidenziato ma si fa sì che si scelgano assegnamenti che non solo migliorino in media la distribuzione, ma che provochino il minor numero possibile di aumenti alle somme di probabilità e quindi all'utilizzo di unità funzionali. I contributi negativi verranno quindi considerati solo per ordinare assegnamenti che risulterebbero pari considerando le *self-forces* e i contribuiti positivi delle *successors' and predecessors' forces*.

Ci si può chiedere se questa modifica nel calcolo delle forze provochi il trascurare degli scheduling che effettivamente avrebbero un grosso contributo negativo da parte delle *predecessors' and successors' forces* e che cioè provocherebbero una generale riduzione della concentrazione dell'uso di unità funzionali. Tuttavia se un'operazione dà un contributo negativo ad una qualche

predecessors' and successors' forces, significa che la riduzione corrispondente della mobilità darà comunque vita ad una *self-force* negativa in almeno uno dei passi della finestra temporale ristretta da un altro assegnamento. Se tale assegnamento avesse anche una *predecessors' and successors' force* non positiva o comunque non eccessivamente positiva, tale assegnamento verrebbe comunque prima o poi scelto dall'algoritmo. In tale modo vengono comunque applicati gli effetti benefici previsti da quegli scheduling che si temeva di trascurare. Una volta comunque applicati tali assegnamenti si potrà quindi valutare se l'assegnamento, trascurando gli effetti indiretti provocati da esso (cioè i contributi di predecessori e successori), provochi effettivamente un miglioramento nella concentrazione dell'utilizzo delle unità funzionali e quindi decidere se debba essere comunque eseguito.

Con la modifica evidenziata in pratica si è variata la filosofia della scelta del prossimo scheduling: non più cercare l'assegnamento che migliori mediamente la somma di probabilità relativa al tipo dell'operazione che si assegna e le somme di probabilità relative ai tipi dei suoi predecessori e dei suoi successori, ma cercare l'assegnamento che migliori la somma relativa all'operazione dell'assegnamento e che non peggiori o peggiori il meno possibile quelle relative allo stesso tipo negli altri passi di controllo e quelle relative agli altri tipi di operazioni coinvolti indirettamente nell'assegnamento. Questa modifica si farà sentire maggiormente nel caso vengano imposti dei vincoli sulle unità funzionali (4.2).

4.1.4 Scelta del prossimo assegnamento da effettuare

Nella sua formulazione originale l'algoritmo prevede che ad ogni iterazione venga scelto quale operazione schedulare e in quale passo di controllo sulla base delle forze relative alle coppie <operazione-passo di controllo>. In 4.1.1 è stato proposto un criterio tramite il quale fornire un ordinamento parziale delle operazioni. Questo ordinamento parziale è utilizzato per limitare le possibili candidate ad essere scelte come prossima operazione da schedulare per limitare la complessità dell'algoritmo e come è stato mostrato esso non incide particolar-

mente sui risultati ottenuti perchè gli assegnamenti di operazioni appartenenti ad un gruppo hanno poca influenza sulle operazioni degli altri gruppi.

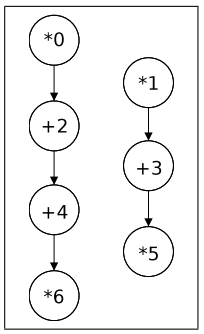
Al contrario all'interno di uno stesso gruppo l'assegnamento di un'operazione incide fortemente sulle altre perchè spesso ne limita le mobilità restringendo le finestre temporali dei loro possibili assegnamenti o modifica i valori delle somme di probabilità. E' quindi cruciale l'ordine in cui all'interno del gruppo vengano scelti gli assegnamenti: la versione originale dall'algoritmo prevede di scegliere l'operazione avente forza minore perchè essa provoca in media una riduzione della richiesta di utilizzo delle risorse necessarie.

Come sottolineato in 3.1.2 (Calcolo delle Forze) tuttavia mentre il calcolo della *self-force* è per così dire esatto, quello delle *predecessors' and successors' forces* è in realtà un'approssimazione basata sul principio di sovrapposizione degli effetti e pertanto non tiene conto esattamente di tutti gli effetti. Ampliando le osservazioni che sono state fatte si può anche evidenziare come lo scheduling di un'operazione provochi degli effetti non solo nelle operazioni dello stesso tipo schedulabili contemporaneamente (calcolato come *self-forces*) e nei suoi predecessori e successori (calcolato tramite le *predecessors' and successors' forces*), ma anche, seppur in maniera minore ed indiretta nei successori e nei predecessori delle operazioni dello stesso tipo contemporanee all'operazione di cui si sta calcolando la forza. Infatti tendenzialmente queste ultime, se l'assegnamento teorizzato verrà effettuato, avranno meno possibilità di essere schedulate nello stesso passo di controllo e questo cambiamento si ripercuoterà anche sui loro predecessori e successori. Questa caratteristica che viene giustamente trascurata per non appesantire la complessità dell'algoritmo, diverrà molto più significativa una volta introdotti dei vincoli sulle risorse come mostrato in 4.2. Infatti nel caso si abbiano a disposizione un numero finito di risorse, un assegnamento che utilizzasse l'ultima risorsa disponibile impedirebbe di fatto lo scheduling di ulteriori operazioni dello stesso tipo in quel passo di controllo modificando quindi la mobilità loro e dei loro predecessori e successori. Queste considerazioni potrebbero far pensare di dare un peso diverso alle due componenti della forza, analogamente a quanto è stato fatto in 4.1.3. In questo modo però si perderebbero ulteriori dati che pur se approssimati possono essere significativi. Invece quello che si vuole è sottolineare che i valori delle forze sono delle

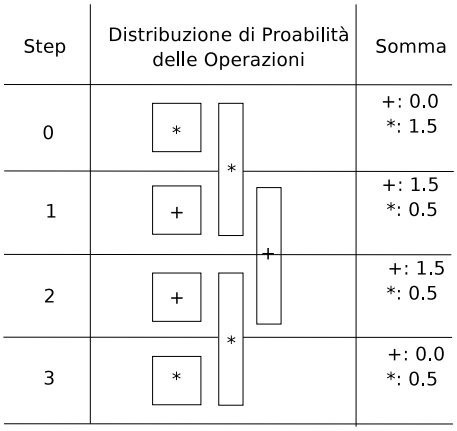
approssimazioni più o meno accurate degli effetti di un assegnamento.

Un'ulteriore considerazione è che operazioni ad elevata mobilità e di un tipo poco presente in alcuni passi di controllo possono avere un valore di *self-force* uguale in più passi di controllo. Per esempio se si considerano un numero elevato di operazioni dello stesso tipo, la prima avente mobilità da 0 a 4, le altre limitate al passo di controllo 2 e se non sono presenti altre operazioni di quel tipo, la prima operazione avrà *self-force* uguale nei passi di controllo fra 0 e 4, 2 escluso, e tale valore sarà relativamente basso. Questo valore potrebbe far sì che questa operazione sia la prima ad essere schedulata in virtù della *self-force* così elevata tale da assorbire eventuali *successors' and predecessors' forces* positive. A quale passo essa verrebbe assegnata dipenderebbe quindi proprio da queste *successors' and predecessors' forces* che però come si è visto sono approssimate. Tale scelta però potrebbe essere cruciale perchè potrebbe influenzare tutti i successivi assegnamenti: infatti mano mano che gli assegnamenti vengono effettuati, la mobilità delle rimanenti operazioni si riduce finchè molte rimangono con una moblità unitaria e vengono quindi automaticamente assegnate.

Come mostra l'esempio riportato in figura 4.3 esistono problemi con operazioni che sin dalla prima iterazione possono avere forze solo positive. Questa caratteristica comporta che tali operazioni verranno schedulate tendenzialmente per ultime. Ciò può comportare che al momento stesso dello scheduling non sia più possibile scegliere in quale passo effettuare l'assegnamento a causa della mobilità ridotta dai precedenti assegnamenti. Tuttavia è facile osservare che queste operazioni possono essere considerate fra le più critiche in quanto qualsiasi scelta viene fatta per il loro assegnamento comporta un peggioramento della distribuzione delle operazioni. Verrebbe quindi da pensare che esse dovrebbero avere precedenza nello scheduling. Un modo per fare ciò potrebbe essere quello di scegliere come prossima operazione da schedulare non quella avente forza minore, ma quella avente la forza migliore più alta dove con forza migliore di un'operazione intendiamo la forza più bassa fra tutte quelle relative agli scheduling di essa possibile. Rimane tuttavia il problema posto inizialmente: le forze come è stato mostrato sono frutto di un calcolo approssimato e quindi alla minor forza, specie se essa si discosta poco dalla seconda più bassa, non necessariamente corrisponde una scelta più vantaggiosa. Sbagliare



(a) DFG di esempio



(b) Somme di Probabilità

Op	CS	SF	OF	TF
*1	0	+0.5	-0.5	0.0
*1	1	-0.5	+0.5	0.0
+3	**1**	0.0	+0.5	**+0.5**
+3	**2**	0.0	+ 0.5	**+0.5**
*5	2	-0.5	+0.5	0.0
*5	3	+0.5	-0.5	0.0

(c) Tabella delle forze delle operazioni da schedulare
SF = self forces
OF = predecessors' and successors' forces
TF = total forces

Figura 4.3. Esempio di Operazione (+3) con Forze solo Positive

nell'assegnare queste operazioni che si è visto essere fra le più critiche potrebbe risultare ancora più dannoso.

Alla luce di queste considerazioni si è quindi affrontato il problema da un altro punto di vista: le scelte compiute dell'algoritmo non devono essere tanto nello stabilire quale operazione debba essere schedulata per prima, ma data un'operazione a quale passo di controllo fra quelli possibili essa debba essere assegnata. La decisione che è stata presa è stata quindi quella di riformulare la scelta del prossimo assegnamento seguendo questo procedimento:

1. si calcola la forza media di ogni operazione ottenuta semplicemente sommando le forze relative a tutti i passi di controllo possibili e dividendo il risultato per il numero stesso dei passi;

2. si calcola lo scostamento della forza minore dell'operazione dalla forza media;

3. si sceglie di schedulare come prossima operazione quella che ha lo scostamento più alto; l'assegnamento viene fatto tradizionalmente nel passo di controllo che ha la forza minore.

In questo modo in un certo senso si cerca di prendere la coppia <operazione-passo di controllo> che ha un margine di guadagno maggiore rispetto alle altre coppie formate con la stessa operazione. Questo consente tendenzialmente di rendere trascurabili le approssimazioni nel calcolo delle forze perchè si considera il caso in cui esse hanno meno rilevanza in termini percentuali. Un'altra possibile scelta sarebbe stata quella di selezionare l'operazione che ha maggiore differenza fra la minor forza e la seconda minor forza, ma in questo caso sarebbero state maggiori le possibilità che le operazioni che sono state definite critiche venissero trascurate almeno inizialmente.

4.1.5 Gestione dei cicli

La gestione dei costrutti iterativi presentata in questo lavoro di tesi differisce da quella proposta da Paulin e Knight. Infatti nell'analisi del SDG l'algoritmo proposto trascura di considerare gli archi di feedback e quindi le relative

dipendenze. Questo corrisponde a schedulare una traccia di esecuzione in cui il corpo dei diversi cicli viene eseguito un'unica volta. Lo scheduling delle operazioni appartenenti alle iterazioni successive rispecchierà le distanze temporali relative fra le operazioni appartenenti alla prima iterazione e verrà implicitamente lasciato agli stadi successivi o al controllore stesso. Il modo più semplice per assegnare le operazioni delle iterazioni successive è schedularle consecutivamente: l'esecuzione dell'iterazione successiva inizia quando è terminata quella precedente. Per migliorare il tempo di esecuzione complessivo in caso di più iterazioni del ciclo è necessario verificare se l'esecuzione di iterazioni successive può essere sovrapposta. Questo può essere delegato a moduli per l'ottimizzazione dello scheduling dei cicli o al Force Directed stesso tramite la replicazione delle operazioni delle diverse iterazioni come mostrato in 3.2.1 (Scheduling con cicli nella versione originale).

Le differenze dell'approccio proposto rispetto a quello originale sono:

- non è necessario imporre dei vincoli relativi ai singoli cicli nella descrizione comportamentale;
- il tempo di esecuzione di una singola iterazione è mediamente maggiore a parità di risorse allocate;
- è possibile schedulare operazioni non appartenenti al corpo del ciclo contemporaneamente a operazioni ad esso appartenenti.

Un modo possibile per forzare un vincolo sulla durata di un'iterazione è manipolare i dati in ingresso all'algoritmo; in particolare è necessario modificare le finestre temporali delle operazioni appartenenti al ciclo in modo tale che la differenza fra l'ASAP dell'operazione corrispondente al costrutto iterativo incrementato del tempo di esecuzione dell'operazione stessa (il primo passo di controllo a cui è possibile assegnare un'operazione appartenente al ciclo) e il maggiore fra i valori ottenuti sommando l'ALAP di un'operazione appartenente al ciclo con il suo tempo di esecuzione (primo passo di controllo in cui sicuramente si può assegnare un'operazione appartenente alla successiva iterazione del ciclo) sia pari al tempo imposto dal vincolo.

4.1.6 Complessità dell'Algoritmo modificato

La complessità dell'algoritmo può essere calcolata come mostrato in 3.1.4, metodo che viene qui ripreso brevemente sottolineando l'unica differenza introdotta; con n è indicato il numero di operazioni del problema, con c il numero di passi di controllo previsto, con B il numero di Blocchi Basici e con b il numero medio di operazioni presente in un blocco basico (quindi $b \cdot B = n$):

1. l'algoritmo viene eseguito al massimo per n iterazioni;
2. ad ogni iterazione si calcolano le forze relative a massimo b operazioni;
3. ogni operazione si può schedulare in massimo c passi di controllo;
4. ogni operazione ha al massimo n fra predecessori e successori, ciascuna schedulabile in massimo c passi di controllo.

Il risultato di questo calcolo porta ad una complessità complessiva di $O(c^2n^2b)$ equivalente a $(O\frac{c^2n^3}{B})$. La riduzione di complessità rispetto alla versione originale è quindi pari ad un fattore B, e quindi nel caso di *SDG* privi di operazioni condizionali il guadagno sarà nullo. L'origine di questa riduzione è dovuta esclusivamente alla modifica mostrata in 4.1.1 (Priorità delle operazioni).

4.2 Estensione all'algoritmo con introduzione dei vincoli sulle risorse

In questa sezione verrà illustrato l'estensione dell'Algoritmo con l'introduzione dei vincoli sulle risorse, ma a differenza di quanto suggerito in [PK87] e [PK89] e riassunto in 3.2.5 non verranno allentati i vincoli temporali nel fare ciò. Ci si può domandare quale sia lo scopo di applicare un algoritmo per minimizzare il numero di risorse quando esse siano già state allocate in un numero prefissato. I motivi possono essere molteplici:

- verificare che non vi siano unità funzionali che sono inutilizzate e che quindi potrebbero essere eliminate;

- nel caso i vincoli sulle risorse si riferiscano solo ad alcuni tipi di unità funzionali, minimizzare il numero di risorse non vincolate;
- cercare di minimizzare il numero di interconnessioni o il numero di registri;
- cercare di uniformare l'utilizzo delle unità funzionali nei vari passi di controllo;
- fornire non solo un possibile scheduling, ma anche fare il binding di operazioni che possono essere eseguite da più tipi di unità funzionali su uno particolare di questi tipi.

L'ultima motivazione anticipa una delle particolarità dell'estensione all'algoritmo presentata in questa sezione e che verrà precisata in 4.2.2.

Le principali modifiche all'algoritmo consistono in una nuova formulazione della distribuzione di probabilità e conseguentemente delle forze e nell'introduzione del *Backtracking*.

4.2.1 Introduzione dei vincoli

I vincoli introdotti si riferiscono al limitare il numero di unità funzionali di un certo tipo allocabili. L'algoritmo non è in grado di rilevare che non esista un possibile scheduling che soddisfi sia il vincolo temporale, sia i vincoli sulle risorse, cioè non è in grado di stabilire se il problema proposto abbia effettivamente soluzione, quindi si devono fornire dei vincoli che permettano almeno uno scheduling ammissibile. E' possibile che i vincoli si riferiscano solo ad alcuni tipi di unità funzionali, ma per applicare concretamente l'algoritmo proposto è necessario fissare dei vincoli per tutti i tipi di unità funzionali come verrà mostrato successivamente. Poichè il numero delle unità funzionali non vincolate sarebbe teoricamente infinito, questo deve essere quantificato in un numero sufficientemente grande: arbitrariamente si è scelto di porlo uguale al numero delle operazioni presenti nel SDG da schedulare. Un suggerimento a come modellizzare il problema è fornito da 3.2.3 (Integrazioni di informazioni relative all'architettura). I pesi da utilizzare non saranno determinati dal costo

delle unità funzionali bensì dal numero di esse disponibili. Per ottenerli basta considerare che il peso presente nel calcolo della forza si riferisca esplicitamente alla somma di probabilità cioè utilizzare la formula:

$$Force(i) = weighted\_DG(i) \cdot x(i) \tag{4.2}$$

dove

Force(i) è il contributo alla forza dell'assegnamento relativo all'i-esimo passo di controllo;

weighted_DG(i) è la somma di probabilità del tipo di unità funzionale che può eseguire l'operazione nell'i-esimo passo di controllo pesata da un opportuno valore;

x(i) è la variazione della somma di probabilità nell'i-esimo passo di controllo a seguito dell'assegnamento.

Per ottenere il valore da attribuire ai pesi basta imporre la condizione per tutte le unità funzionali che la somma di probabilità pesata sia pari ad uno nel caso la somma di probabilità corrisponda esattamente al numero di unità funzionali disponibili di quel particolare tipo. E' proprio per applicare questa uguaglianza che è necessario indicare un numero di unità funzionali anche per i tipi non vincolati in modo tale da poter pesare anche le somme di probabilità relative a questi tipi di unità.

Si considerino due operazioni di tipo diverso, per esempio una somma e una moltiplicazione, e si supponga di avere a disposizione tre sommatori e un moltiplicatore. Si ipotizzi inoltre che le operazioni abbiano la stessa finestra temporale, mobilità pari a due e che nel primo dei due passi di controllo in cui entrambe possono venir schedulate, le somme di probabilità pesate relative ad addizionatori e moltiplicatori siano pari ad uno. Se venisse applicata la formula non pesata della forza e si calcolassero le *self-forces* relative al secondo dei due passi di controllo, il contributo ad esse dato dalla riduzione di probabilità sul primo passo di controllo sarebbe per entrambe -0.5. Si può notare che tale riduzione libererebbe per entrambi i tipi di unità funzionale l'equivalente di 0.5 unità (si è sempre in un ambito probabilistico, quindi ha senso

parlare di frazioni di unità funzionale) e quindi sembrerebbe coerente che i due contributi alle forze siano pari. Questi valori uguali rispecchiano però la concezione classica dell'algoritmo. Infatti se si considerano i vincoli sulle unità funzionali ci si accorge che le due riduzioni di finestra temporale liberano rispettivamente $\frac{1}{6}$ e $\frac{1}{2}$ del totale delle unità funzionali. Poichè è fondamentale che i vincoli vengano rispettati appare chiaro che la riduzione della seconda operazione dovrebbe dare un contributo diverso e più pesante della prima (in questo caso dovrebbe avere un valore minore, perchè le forze migliori sono quelle minori). Per imporre questo comportamento è sufficiente utilizzare non la formula classica per il calcolo della forza, ma la versione proposta in 4.1 con l'accortezza di pesare la forza non solo pesando il primo termine, cioè la somma di distribuzione di probabilità ma anche il secondo cioè la differenza di somma di probabilità. Ovvero:

$$x(i) = \triangle weighted\_DG(i) = \triangle DG(i) \cdot weight(fu) \tag{4.3}$$

dove $weight(fu)$ è il peso relativo all'unità funzionale considerata. Applicando questa formula si otterranno come contribuiti i valori di -0.16 per l'addizione e di -0.50 per la moltiplicazione.

In generale si può notare come l'algoritmo per soddisfare i vincoli relativi alle risorse debba far sì che le somme di probabilità pesate di tutti i passi di controllo e per tutte le unità funzionali debbano avere un valore inferiore all'unità ovvero che sia inferiore ad essa il valore massimo assunto da una somma di probabilità. Questo obbiettivo rientra appieno nella filosofia iniziale del Force Directed.

Alla luce della formula generale appena introdotta

$$Force(i) = weighted\_DG(i) \cdot \triangle DG(i) \cdot weight(fu) \tag{4.4}$$

è possibile fornire una riformulazione della formula stessa. Tale riformulazione in pratica non cambierà il valore dei termini presenti nella formula e quindi il suo risultato, ma cambierà il loro significato fornendo una nuova interpretazione al problema della minimizzazione delle risorse.

Fino ad ora le distribuzioni di probabilità descritte nelle sezioni precedenti hanno sempre fatto riferimento alla probabilità che una certa operazione venisse schedulata in un certo passo di controllo e le somme di probabilità sono state riferite al numero di operazioni di un certo tipo che probabilmente sarebbero state assegnate a quel determinato passo di controllo. Con questa nuova formulazione al concetto di somma di probabilità viene sostituito quello di *percentuale di occupazione*. Si associa un valore di questo tipo ad ogni coppia <passo di controllo-tipo di unità funzionale>: questo valore indicherà la percentuale di unità funzionali di quel tipo che probabilmente (nello stesso senso inteso nella precedente formulazione; la probabilità diventa certezza allorchè tutte le operazioni sono state assegnate) sarà occupata in quel passo di controllo. Similmente alla interpretazione originaria si suppone che se un'operazione ha una mobilità fissata n, tale operazione occuperà probabilmente una frazione di unità funzionale pari a 1/n per tutti i passi di controllo della sua finestra temporale. Per ottenere la percentuale di occupazione non di una singola unità funzionale ma dell'intero blocco è sufficiente dividere il valore ottenuto per il numero stesso delle unità funzionale di quel particolare tipo. Sommando le occupazioni relative alle singole operazioni ed applicando le considerazioni sui costrutti condizionali presentate in 4.1 si otterranno facilmente le percentuali di occupazione. A questo punto per ottenere i contributi alle forze sarà sufficiente applicare:

$$Force(i) = \%occ(i,F) \cdot \triangle\%occ(i,F) \tag{4.5}$$

dove

Force(i) è il contributo alla forza dell'assegnamento relativo all'i-esimo passo di controllo;

%occ(i,F) è la percentuale di occupazione del tipo di unità funzionale F in quel passo di controllo;

$\triangle$ **%occ(i,F)** è la variazione di occupazione a seguito dello scheduling.

Si può facilmente dimostrare che (4.4) e (4.5) sono equivalenti cioè che il contributo alla forza calcolato è identico. Per dimostrare ciò vengono introdotti alcuni nuovi simboli:

o una generica operazione di un tipo fissato eseguibile dalla risorsa F e schedulabile in un passo di controllo i;

prob(o,i) è la probabilità che l'operazione o venga schedulata nel passo di controllo i;

occ(o,i,F) è la percentuale di occupazione di unità funzionali di tipo F da parte dell'operazione o nel passo di controllo i;

fu(F) è il numero di unità funzionali di tipo F.

Alcune premesse:

$$\frac{1}{mobilita'(o)} = \frac{1}{mobilita'(o)}$$

da questa identità segue

$$prob(o,i) = fu(F) \cdot \frac{1}{mobilita'(o) \cdot fu(F)} \tag{4.6}$$

$$prob(o,i) = fu(F) \cdot occ(o,i,F)$$

$$\sum_o prob(o,i,F) = fu(F) \sum_o occ(o,i,F)$$

$$DG(i) = fu(F) \cdot \%occ(i,F)$$

$$\frac{DG(i)}{fu(F)} = \%occ(i,F) \tag{4.7}$$

e per come sono stati ricavati i pesi:

$$weight(F) = \frac{1}{fu(F)} \tag{4.8}$$

Si può quindi dimostrare che:

$$\%occ(i,F) \cdot \triangle\%occ(i,F) = Force(i) = weighted\_DG(i) \cdot \triangle DG(i) \cdot weight(F)$$
$$\%occ(i,F) \cdot \triangle\%occ(i,F) = Force(i) = DG(i) \cdot weight(F) \cdot \triangle DG(i) \cdot weight(F)$$

applicando la sostituzione evidenziata nell'uguaglianza (4.8) si ottiene

$$\%occ(i,F) \cdot \triangle\%occ(i,F) = Force(i) = DG(i) \cdot \frac{1}{fu(F)} \cdot \triangle DG(i) \cdot \frac{1}{fu(F)}$$

quindi applicando l'uguaglianza (4.7) si ottiene

$$\frac{DG(i)}{fu(F)} \cdot \triangle \frac{DG(i)}{fu(F)} = Force(i) = \frac{DG(i) \cdot \triangle DG(i)}{(fu(F))^2}$$

e poichè fu(F) è di fatto una costante si ottiene l'uguaglianza che dimostra l'equivalenza formule per il calcolo dei contributi delle forze e quindi si dimostra l'equivalenza delle due formulazioni del problema

$$\frac{DG(i) \cdot \triangle DG(i)}{(fu(F))^2} = Force(i) = \frac{DG(i) \cdot \triangle DG(i)}{(fu(F))^2} \tag{4.9}$$

4.2.2 Introduzione del binding su un tipo di unità funzionale

Nella formulazione stessa del *force Directed scheduling* è implicito che oltre ad eseguire uno scheduling l'algoritmo effettua anche un'allocazione delle unità funzionali necessarie ad eseguire lo scheduling stesso. Come si è visto in 4.2 nelle motivazioni addotte per introdurre i vincoli tecnologici nell'algoritmo, i vincoli più o meno parziali in effetti non limitano la funzione di allocazione dell'algoritmo stesso. Per quanto riguarda il binding finora non è stato consid-

erato nella descrizione dell'algoritmo in quanto quest'ultimo non fa differenza fra le singole istanze di uno stesso tipo di unità funzionale.

I vincoli tecnologici introdotti però potrebbero prevedere l'esistenza di più tipi di unità funzionali che possano eseguire uno stesso tipo di operazione. Solitamente tali tipi di unità funzionali si differenziano per qualche parametro che influisce sulle metriche complessive, come ad esempio il tempo di esecuzione o la potenza dissipata nell'eseguire l'operazione. Nella formulazione originaria dell'algoritmo non essendo contemplati limitazioni al numero di unità funzionali, l'esistenza di più unità funzionali che possono eseguire lo stesso tipo di operazione non complica il problema dello scheduling. Infatti basterà selezionare fra i tipi di unità funzionali quello che risulterà migliore secondo una certa metrica ed assegnare ad esso le operazioni in oggetto. Se si ipotizza però di avere i vincoli sul numero di unità funzionali allocabili questa semplificazione non è più possibile perchè potrebbero essere necessari per ottenere uno scheduling valido entrambi i tipi di unità funzionale. Ci si può chiedere perchè si introducano in un'architettura almeno due tipi diversi di unità che eseguano la stessa operazione. Le motivazioni possono essere diverse:

- uno dei due tipi di unità viene allocato perchè consente di eseguire anche altre operazioni: per esempio sommatori e ALU;
- uno dei due tipi di unità ha prestazioni (tempo di esecuzione) migliori dell'altra, ma costi più elevati; si è scelto quindi nello stabilire l'architettura un compromesso fra l'utilizzo esclusivo di un tipo di unità funzionale e fra l'utilizzo esclusivo dell'altro tipo.

Questa possibilità pone un nuovo problema: quale criterio utilizzare per assegnare un'operazione ad una unità funzionale di un tipo piuttosto che ad una di un altro qualora questa sia eseguibile da entrambi. Una prima ipotesi è quella di utilizzare un altro algoritmo per prendere queste decisioni; una volta ottenuto i risultati basterebbe eseguire il Force Directed costruendo le somme di probabilità o l'occupazione delle unità funzionali sulla base degli assegnamenti <operazione-tipo di unità funzionale> già eseguiti. Questi assegnamenti in realtà influenzano direttamente lo scheduling perchè i tempi di esecuzione di un'operazione potrebbero dipendere dal tipo di unità funzionale su cui essa

viene eseguita. Questa diversità si ripercuote sulla mobilità dell'operazione e quindi indirettamente anche sulla mobilità di successori e predecessori. Sulla base di queste considerazioni si è scelto di incorporare all'interno dell'algoritmo il binding di un'operazione ad un tipo di unità funzionale (non alla specifica istanza dell'unità funzionale).

Per modellizzare questo problema, viene esteso il concetto di forza: viene infatti calcolata una forza per ogni assegnamento possibile che, sulla base di quanto esposto, sarà costituito non dalla coppia <operazione-passo di controllo> bensì dalla terna <operazione-passo di controllo-tipo di unità funzionale>. Di fatto vi sarà differenza solo per le operazioni che possono essere eseguite da più tipi di unità. Per le altre operazioni esisterà un'unica forza per la coppia <operazione-passo di controllo> e il binding sarà automaticamente fatto con l'unico tipo di unità funzionale possibile. Il calcolo della forza viene fatto utilizzando la formula già proposta in 4.2.1:

$$Force(i) = \%occ(i,f) \cdot \triangle\%occ(i,f) \tag{4.10}$$

cioè utilizzando non la somma di probabilità bensì la percentuale di occupazione di quel tipo di unità funzionale.

Come verrà mostrato tuttavia tale formula nel caso in oggetto non costituisce una reinterpretazione della formula originale con l'introduzione dei pesi poichè le $\%occ$ verranno calcolate in modo diverso, quindi l'uguaglianza (4.9) non sarà valida in questo caso. Per determinare le $\%occ(i)$ è necessario calcolare i contributi relativi alle singoli operazioni. Essi dipendono dalla probabilità che l'operazione venga schedulata in quel passo di controllo su quel tipo di unità funzionale. In questo caso la funzione di distribuzione della probabilità sarà bivariata: si deve assegnare una probabilità per ogni coppia <tipo di unità funzionale-passo di controllo>. Tale funzione sarà ovviamente nulla per le coppie che non consentono un assegnamento valido. Una caratteristica che tale funzione deve necessariamente avere è:

$$\sum_f f(c,f,o) = \frac{1}{mobilita'(o)} \tag{4.11}$$

dove *c* è un generico passo di controllo appartenente alla finestra temporale dell'operazione, *f* è un generico tipo di unità funzionale ed *o* è l'operazione. Sostanzialmente cioè la probabilità che un'operazione venga assegnata ad un passo di controllo possibile deve essere uguale per tutti i passi di controllo possibili (come nel caso in 3.1.1 - calcolo della probabilità nell'algoritmo originale).

Analogamente alla considerazioni fatte per scegliere la funzione di distribuzione della probabilità nel caso monovariato riportate in 3.1.1 anche in questo caso alla variabile aleatoria indicante l'assegnamento dell'operazione verrà assegnata una distribuzione di probabilità uniforme ma a differenza del caso originale essa sarà bivariata. Tuttavia la seconda variabile sostegno della funzione non sarà il tipo di unità funzionale, ma le singole istanze di unità funzionali presenti. Cioè si ipotizza che la probabilità che una certa operazione sia eseguita in un certo passo di controllo su una unità funzionale che la possa eseguire sia uguale per tutte le unità funzionale. Se si fosse utilizzata come seconda variabile il tipo di unità, si avrebbe avuto che un tipo di unità funzionale di cui sono state istanziate tre esemplari avrebbe avuto la stessa possibilità di schedulare un'operazione di un tipo istanziato in un solo esemplare. Si è quindi ottenuto

$$\mathbb{P}(c, fn, 0) = \frac{1}{mobilita'} \cdot \frac{1}{n(o)} \tag{4.12}$$

(*fn* è l'n-esima unità funzionale allocata che può eseguire l'operazione, *n(o)* è il numero di unità funzionali che possono eseguire l'operazione *o*)

A partire da questa distribuzione di probabilità si costruisce la densità di probabilità bivariata con variabili passo di controllo e tipo di unità funzionali. Per ottenere la probabilità che un tipo di unità possa eseguire l'operazione è sufficiente sommare le probabilità relative alle sue singole istanze:

$$\mathbb{P}(c, f, o) = \sum_{fn \epsilon f} \mathbb{P}(c, fn, o) \tag{4.13}$$

Si ricava facilmente che tale probabilità, detto $fu(f)$ il numero di unità funzionale di tipo *f* è esprimibile in questi termini:

$$\mathbb{P}(c,f,o) = \sum_{fn \epsilon f} \mathbb{P}(c,fn,o) = fu(f) \cdot \mathbb{P}(c,fn,o) = \frac{fu(f)}{n(o)} \cdot \frac{1}{mobilita'} \tag{4.14}$$

cioè la probabilità che un'operazione venga assegnata in un passo di controllo ad un tipo di unità funzionale è la probabilità che essa venga assegnata a quel passo di controllo pesata dal rapporto fra le unità funzionali di quel tipo e il numero complessivo di unità funzionali che possono eseguire quell'operazione.

Come si è visto in 3.1.1 dire che una certa operazione è assegnata con una certa probabilità a quel passo di controllo equivale a dire che in media una frazione di unità funzionale pari al valore della probabilità verrà occupata in quel passo di controllo. Per ottenere da questa frazione la percentuale di occupazione di un tipo di unità di controllo è sufficiente suddividere questo valore per il numero di unità funzionali disponibili. Si ha così che:

$$occ(o,i.f) = \frac{fu(f)}{n(o)} \cdot \frac{1}{mobilita'} \cdot \frac{1}{fu(f)} = \frac{1}{mobilita' \cdot n(o)} \tag{4.15}$$

A questo punto è sufficiente applicare

$$\%occ(i.f) = \sum_{o} occ(o,i,f) \tag{4.16}$$

e l'equazione (4.5) per ottenere i contributi alle forze.

Sull'uguaglianza (4.15) è bene fare delle considerazioni: il contributo di un'operazione all'occupazione di una certa unità funzionale dipende solo dal numero totale di unità funzionali che possono eseguirla, non dal numero di unità funzionali dei diversi tipi; questo perchè l'uniformità delle distribuzioni di probabilità fa sì che un'operazione tenda a caricare l'occupazione dei diversi tipi di unità funzionali in modo proporzionale. Un secondo aspetto da tenere in considerazione è che se esiste un solo tipo di risorsa che può eseguire un'operazione il numero di unità totali corrisponderà al numero di unità di questo tipo e quindi per tutti questi tipi di unità funzionali, per esempio per quelle di tipo F poiche $n(o) = fu(F)$:

$$
\begin{aligned}
\%occ(i,F) &= \sum_o occ(o,i,F) \\
&= \sum_o \frac{1}{mobilita'(o) \cdot n(o)} \\
&= \sum_o \frac{1}{mobilita'(o) \cdot fu(F)} \\
&= \frac{1}{fu(F)} \cdot \sum_o \frac{1}{mobilita'(o)} \\
&= \frac{1}{fu(F)} \cdot DG(i) \\
&= weighted\_DG(i)
\end{aligned} \tag{4.17}
$$

L'ultima formulazione del problema presentata è applicabile quindi anche ai casi considerati in 4.2.1 e in 3.1. In quest'ultimo caso è sufficiente ipotizzare che tutti i tipi di unità funzionali siano state istanziate in un numero uguale ed arbitrario. In questo modo le forze calcolate utilizzando le formule (4.15), (4.16) e (4.5) sono uguali a quelle calcolate con la formula tradizionale a meno di una costante pari all'inverso del quadrato del numero arbitrario. Quindi è possibile realizzare un'unica versione dell'algoritmo utilizzabile in presenza o meno di vincoli sul numero di risorse ed operazioni eseguibili da più tipi di unità funzionali. Ci si può chiedere se anche questa formulazione sia equivalente alle due precedenti. La risposta è no: in realtà il caso senza vincoli sul numero di risorse costituisce un caso particolare del caso con i vincoli, ma non viceversa. Infatti questo caso non costituisce una semplice reinterpretazione del caso base per quanto mostrato nell'uguaglianza (4.17). L'uguaglianza vale per l'ipotesi $n(o) = fu(F)$, condizione vera solo se non vi sono operazioni eseguibili da tipi diversi di unità funzionali. In caso contrario l'occupazione di un tipo di risorsa ($\%occ(i,F)$) non è equivalente ad una somma di probabilità pesata perchè non vi è un unico peso identico per tutta la somma, ma i contributi dovuti alle singole operazioni vengono pesati con un valore tipico del tipo di operazione

($\frac{1}{n(o)}$)che quindi può variare da operazione a operazione:

$$\%occ(i,F) = \sum_{o} \frac{1}{mobilita' \cdot n(o)} \tag{4.18}$$

4.2.3 Introduzione del Backtracking

Finora si è ipotizzato che l'algoritmo effettui un nuovo assegnamento ad ogni iterazione fino a schedulare tutte le operazioni. Tuttavia il Force Directed scheduling non è un algoritmo esatto: il numero di risorse allocato da esso potrebbe non corrispondere all'effettivo numero minimo di risorse sufficiente a schedulare correttamente il SDG. Allo stesso modo è possibile che la prima soluzione prodotta dall'algoritmo modificato non rispetti i vincoli perchè potrebbe utilizzare per qualche risorsa un numero di unità maggiore di quello minimo che potrebbe e se il vincolo sul numero di unità di questo tipo coincidesse proprio con il minimo, tale vincolo verrebbe violato.

E' quindi necessario che l'algoritmo scarti le soluzioni non rispettose dei vincoli e ne cerchi di nuove fino ad ottenerne una valida. Per fare ciò è stata inserita all'interno di esso la tecnica di *backtracking* (sulle tecniche di backtracking cft 2.3). In particolare il problema dello scheduling viene modellizzato sottoforma di albero di ricerca di grado determinato dai dati del problema. Ogni arco dell'albero rappresenta un assegnamento operazione-unità funzionale-passo di controllo, ogni nodo uno scheduling parziale formato dagli assegnamenti corrispondenti agli archi che congiungono quel nodo alla radice dell'albero, ogni foglia uno scheduling completo. Il percorso che l'algoritmo compie lungo l'albero a partire dalla radice scendendo verso le foglie è determinato quindi dalla scelte fatte ad ogni iterazione. Se l'algoritmo giunge in una foglia corrispondente ad una soluzione non accettabile, esso dovrebbe risalire al nodo precedente e compiere una nuova scelta fra quelle non ancora effettuate. Se non vi fossero più scelte disponibili dovrebbe risalire di un ulteriore livello e così via.

Questa appena fornita è però la versione più semplicistica della tecnica di backtracking utilizzabile. Infatti diversi miglioramenti ad essa possono essere introdotti e verranno esposti qui di seguito.

4.2.3.1 Taglio di sottoalberi esplicitamente "morti" o "moribondi"

Durante la discesa lungo l'albero è possibile verificare ad ogni nodo, cioè ad ogni aggiunta di un assegnamento allo scheduling parziale costruito fino a quel punto, se lo scheduling corrispondente, seppur incompleto, già violi qualcuno dei vincoli. In tal caso il sottoalbero avente origine da quel nodo avrà come foglie solo soluzioni non ammissibili in quanto in nessun modo gli assegnamenti successivi potrebbero eliminare la violazione del vincolo. Risulta quindi inutile continuare la ricerca lungo quella strada: l'algoritmo annulla l'ultimo assegnamento effettuato (risalendo l'albero di un livello), quindi sceglie il prossimo miglior assegnamento a partire da quel nodo e lo effettua.

Un ulteriore miglioramento delle prestazioni si può ottenere anticipando il taglio del sottoalbero: nel momento in cui viene calcolata la forza relativa all'assegnamento che provocherebbe la violazione dei vincoli si calcola la variazione di occupazione delle unità funzionali. Parallelamente a questo calcolo e senza aumentare la complessità totale dell'algoritmo è anche possibile calcolare la variazione dell'occupazione effettiva delle unità funzionali. Con il termine occupazione effettiva si intende la percentuale di occupazione delle unità funzionali dovuta non a parte di operazioni che probabilmente verranno assegnate in questo passo di controllo, ma ad operazioni che saranno sicuramente assegnate (a meno di successivi passi di backtracking) a questo passo di controllo e a questo tipo di unità funzionali perchè relative ad assegnamenti già fatti nelle iterazioni precedenti. Qualora l'assegnamento relativo alla terna <operazione-passo di controllo-unità funzionale> facesse superare all'occupazione effettiva del tipo di unità funzionale il valore di uno, esso provocherebbe una palese violazione dei vincoli. Quindi è inutile consentire questo assegnamento visto che sarebbe cancellato nel momento in cui esso venisse attuato; pertanto si pone per convenzione ad un valore arbitrariamente elevato la forza relativa a tale tipo di assegnamento. Tali forze verranno indicate nel proseguo del lavoro come forze bloccate.

Ci si può chiedere se si possano fare considerazioni similari considerando non solo l'operazione in oggetto ma anche predecessori e successori: questa verifica sul fatto che l'assegnamento di un'operazione possa rendere impossi-

bile l'assegnamento di un suo successore o predecessore è possibile ed è semplice fare solo qualora il predecessore o il successore veda la sua mobilità ridotta ad uno. In tal caso è sufficiente controllare (considerando ovviamente le particolarità introdotte dai costrutti condizionali) se esista ancora un'unità funzionale libera capace di eseguire l'operazione. In caso ciò non si verificasse anche in questo caso la forza dell'assegnamento in esame verrà posta ad un valore arbitrariamente grande.

E' da sottolineare come lo schedulare le operazioni a gruppi, a seconda dell'appartenenza ad un blocco basico, possa portare alla luce più precocemente violazioni dei vincoli sulle risorse poichè vengono schedulate a breve distanza le operazioni appartenenti ad un certo blocco basico che per questo motivo avranno le finestre temporale maggiormente sovrapposte. Anticipare la rilevazione delle violazioni riduce i passi di backtracking da compiere e quindi anche il tempo di esecuzione dell'algoritmo.

4.2.3.2 Taglio di sottoalberi implicitamente morti

E' facile dimostrare essendo il Force Directed scheduling un algoritmo costruttivo che se uno scheduling parziale non soddisfa uno o più vincoli, non li soddisferanno neppure gli scheduling parziali che lo contengono (uno scheduling parziale contiene un altro scheduling se tutti gli assegnamenti del secondo sono presenti nel primo). Si identifichino quindi gli scheduling parziali come insieme di assegnamenti e si supponga che l'algoritmo abbia prodotto uno scheduling parziale $S0 = \{a_1, a_2, a_{n-1}\}$ e all'iterazione successiva $S1 = \{a_1, a_2, a_{n-1}, a_n\}$. Sia a_{n+1} il successivo miglior assegnamento e quindi $S2 = S1 \cup \{a_{n+1}\}$ e si supponga che $S2$ non rispetti i vincoli. L'algoritmo quindi cancella l'ultima decisione presa e quindi da $S2$ torna a $S1$ cancellando l'assegnamento a_{n+1} (o con il miglioramento suggerito in 4.2.3.1 nel nodo equivalente a $S1$ blocca direttamente la forza relativa a a_{n+1}), quindi sceglie il secondo miglior assegnamento indicato come b_{n+1} ottenendo un nuovo scheduling parziale $S3 = S1 \cup \{b_{n+1}\}$. $S3$ soddisfa i vincoli quindi l'algoritmo può accettarlo. Se gli assegnamenti a_{n+1} e b_{n+1} non si riferiscono alla stessa operazione nel nodo che rappresenta lo scheduling $S3$ l'assegnamento a_{n+1} teoricamente risulterà come uno di quelli

possibili. Non solo, ci sono anche possibilità che esso risulti come il miglior assegnamento possibile visto che lo era per il nodo $S1$ che differisce da $S3$ per un solo assegnamento. Se venisse effettuato si giungerebbe in un nuovo nodo $S4 = S3 \cup \{a_{n+1}\} = S1 \cup \{a_{n+1}, b_{n+1}\} = S2 \cup \{b_{n+1}\}$. Ma $S2$ è uno scheduling parziale non ammissibile, quindi non lo è neppure $S4$. E' quindi opportuno che nel sottoalbero avente origine da $S1$ venissero tagliati tutti i sottoalberi aventi come arco entrante nel relativo nodo radice un arco contrassegnato con l'assegnamento a_{n+1} perchè conterrebbero solo soluzioni non valide. Nel caso si utilizzasse solo il miglioramento proposto in 4.2.3.1 l'algoritmo dovrebbe comunque ricalcolare la forza in ogni nodo e solo a quel punto si accorgerebbe dell'assegnamento impossibile e la riblocchrebbe. In questo caso quindi il taglio dei sottoalberi aventi come arco in ingresso a_{n+1} coincide semplicemente con l'evitare di dover ricalcolarne la forza. L'assegnamento a_{n+1} però non necessariamente non fa parte di alcuna soluzione valida. Infatti se consideriamo lo scheduling $S5 = S0 \cup \{a_{n+1}\}$, esso a priori non viola alcun vincolo (potrebbe farlo ma per scoprirlo è necessario effettuare i calcoli) poichè $S5$ contiene sì a_{n+1}, ma non contiene $S1$.

Queste considerazioni quindi mostrano come sia possibile ogni volta che si taglia un sottoalbero a causa di una violazione di vincoli, tagliare i sottoalberi appartenenti al sottoalbero generato dal genitore del sottoalbero tagliato i cui archi di ingresso siano contrassegnati allo stesso modo.

Come è stato mostrato i tagli possibili si riferiscono solo alla porzione di albero che ha origine nel nodo corrispondente all'ultimo scheduling valido. E' facile constatare però applicando l'algoritmo che difficilmente un assegnamento che fa scattare una violazione del vincolo appartenga alla soluzione ottimale. Tipicamente infatti perchè esso possa diventare valido devono annullarsi una serie di scelte fatte in precedenza, scelte che comunque al momento della loro effettuazione corrispondevano a forze migliori di quella dell'assegnamento colpevole. Le operazioni vengono assegnate in ordine in base alle loro forze: se tale assegnamento fosse effettivamente molto valido avrebbe avuto una forza migliore, sarebbe stato schedulato prima e sarebbero stati quindi altri gli assegnamenti che avrebbero fatto scattare la violazione di vincolo. Sulla base di queste considerazioni e per evitare di esplorare sottoalberi simili a quelli già

scartati durante qualche esplorazione, gli assegnamenti scartati in qualche punto dell'algoritmo non vengono poi presi in considerazione come se nulla fosse accaduto, ma vengono penalizzati dando prima strada a scelte e quindi percorsi non ancora intrapresi. In un nodo qualsiasi dell'albero quindi la scelta sul prossimo assegnamento non si baserà più solo sulle forze ma anche sulla storia passata dell'esplorazione.

Spesso però, soprattutto nel caso di vincoli molto stretti diversi assegnamenti di operazioni sono strettamente correlati, in quanto un assegnamento può limitare ad uno la mobilità di molti successori o predecessori. Per questo motivo quando si sceglie un assegnamento è possibile che automaticamente ne vengano imposti alcuni relativi ai successori o ai predecessori dell'operazione in oggetto. Poichè questi assegnamenti sono per così dire costretti, il discorso fatto precedentemente sulla possibilità che un assegnamento che provochi la violazione di un vincolo difficilmente appartenga alla soluzione ottimale decade. Questo comporta che al momento di considerare la storia passata dell'esplorazione si tenga presente anche se assegnamenti poi rivelatisi scorretti non fossero ormai costretti in quanto unici possibili per le'operazioni. In tal caso tali assegnamenti continueranno ad essere tenuti in considerazione.

4.2.3.3 Assegnamenti forzati

Durante l'algoritmo è possibile che un'operazione veda ridursi ad uno la propria mobilità ed esista un solo tipo di unità funzionale capace di eseguirla. In tal caso non è necessario aspettare che l'unico assegnamento possibile abbia la miglior forza in assoluto (se poi si utilizza la versione proposta in 4.1 esso avrà differenza di forza peggiore in quanto 0 e perciò verrà schedulato per ultimo). Posticipando questa scelta che è di fatto obbligata, non solo si ha un'overhead nel tempo di esecuzione dell'algoritmo dovuto al ricalcolo della forza corrispondente, ma si rischia di individuare tardivamente la violazione di vincoli causati da quel particolare assegnamento. Questo comporta la necessità di compiere un maggior numero di passi di backtracking che avrebbero potuto essere evitati evidenziando anticipatamente che l'assegnamento, seppur obbligato, era scorretto.

Esistono inoltre delle situazioni di operazioni che, pur se non hanno mobilità unitaria e un solo tipo di unità funzionale a disposizione, si trovano ad avere implicitamente un unico assegnamento possibile. E' il caso di operazioni che abbiano visto alcuni degli assegnamenti teoricamenti possibili scartati perchè risultati appartenere a scheduling non validi oppure abbiano le forze relative a qualche assegnamento ad un livello arbitrariamente alto ad indicazione che tali assegnamenti provocherebbero una violazione di qualche vincolo. Se il numero di assegnamenti rimasti possibili, cioè aventi una forza calcolata e non imposta arbitrariamente, fosse in numero pari ad uno, la situazione sarebbe simile a quella delle operazioni mostrate nel paragrafo precedente e pertanto è utile al fine della riduzione del tempo di computazione dell'algoritmo effettuare il relativo assegnamento immediatamente.

4.2.3.4 Backtracking Anticipato

In 4.2.3.3 è stato mostrato che è possibile che un'operazione ad un certo momento dell'esecuzione dello scheduling si ritrovi in pratica con un unico assegnamento ancora effettuabile che non violi alcun vincolo. C'è anche la possibilità che un'operazione ad una certa iterazione dell'algoritmo non abbia più assegnamenti possibili, anche con mobilità ancora positiva, perchè tutti gli assegnamenti all'interno della finestra temporale provocherebbero una violazione di un qualche vincolo. E' evidente che nessuno scheduling costruito a partire da una soluzione parziale in cui un'operazione non può più essere schedulata sarà una soluzione accettabile. Il perchè possa accadere questa circostanza, che solitamente si manifesta dopo una serie di assegnamenti forzati, è dovuta all'approssimazione mostrata in 3.1.2 nel calcolo delle *predecessors' and successors' forces* che si ripercuote anche nel calcolo della occupazione effettiva mostrata in 4.2.3.1.

Qualora questa condizione si verificasse e poichè sarebbe inutile continuare uno scheduling a partire da quello attuale, cosa che provocherebbe solo assegnamenti inutili che andrebbero comunque poi cancellati, l'algoritmo deve compiere un passo di backtracking. Questo tuttavia non implica che tutte le operazioni tornino ad essere schedulabili in almeno un passo di controllo. E' vero

che in quel particolare nodo dell'albero di ricerca non si sarebbe effettuato un assegnamento a partire dalla condizione raggiunta se vi fosse un'operazione non schedulabile, ma è anche vero che quando si compie un assegnamento e poi immediatamente dopo si effettua un passo di backtracking, il nodo in cui ci si trova è quello precedente le due mosse, ma lo stato in cui si trova l'algoritmo è diverso perchè uno dei possibili assegnamenti in quel nodo è stato scartato. Se questo assegnamento era un assegnamento forzato, l'operazione corrispondente non potrà più essere schedulata e sarà quindi necessario un ulteriore passo risalendo l'albero. Il backtracking continuerà a risalire l'albero fino a percorrere in senso inverso un arco non corrispondente ad un assegnamento forzato. A quel punto ripartirà la costruzione dello scheduling.

4.2.3.5 Grado dell'albero di Ricerca

Il grado di un albero è il numero massimo di figli che un nodo può avere. Analizzando il problema dello scheduling teoricamente l'albero di ricerca che lo rappresenta potrebbe avere un grado pari al prodotto fra il numero di tipi di unità funzionali, il numero di operazioni e il numero di passi di controllo. Infatti nel caso pessimo inizialmente è possibile che qualsiasi operazione possa venir schedulata in qualsiasi passo di controllo su qualsiasi tipo di unità funzionale. Ad ogni assegnamento possibile corrisponde un arco uscente dal nodo radice dell'albero e conseguentemente un nodo figlio. E' ovvio che questo calcolo è troppo conservativo e che nella maggior parte dei casi il grado dell'albero sarà nettamente inferiore, tuttavia esso è comunque troppo elevato.

In realtà si può facilmente intuire che partendo da uno scheduling parziale, se i migliori assegnamenti via via sopravvissuti conducono tutti a una violazione di qualche vincolo e devono essere scartati, è probabile che il problema principale e la causa delle successive violazioni risieda proprio nello scheduling effettuato fino a quel punto. Per ridurre i tempi di computazione medi si può quindi pensare di limitare il numero di tentativi di estensione di uno scheduling parziale. Raggiunto il bonus di tentativi concessi si cancellerà l'ultimo assegnamento anche se non ha provocato alcuna violazione dei vincoli e se ne proverà un altro. Queste affermazioni si traducono sull'albero di ricerca nella limi-

tazione forzata del suo grado. Utilizzando una metafora botanica, se il numero di tentativi viene fissato ad n, ad ogni nodo verranno tagliati tutti i rami verso i figli esclusi gli n ritenuti più robusti.

E' possibile anche se molto poco probabile che in questo modo vengano eliminate dall'albero di ricerca tutte le foglie relative alle soluzioni valide. In questo malaugurato caso quindi l'algoritmo non troverebbe alcuna soluzione. Per ovviare a ciò è sufficiente impostare l'algoritmo in modo tale che, qualora giunga ad esplorare l'albero potato senza trovare alcuna soluzione, riparta da zero costruendo un nuovo albero di ricerca forzando nuovamente un grado, ovviamente maggiore di quello precedente, che tenga anche conto delle scelte già eliminate nell'albero precedente perchè causa di violazioni dei vincoli. Se nuovamente non si trovasse una soluzione valida, è possibile ripetere più volte l'estensione del grado dell'albero.

4.2.4 La scelta del prossimo assegnamento da effettuare

In 4.1.4 è stata proposta una modifica su come effettuare la scelta relativa al prossimo assegnamento da effettuare. La modifica consiste nello scegliere non l'operazione che ha per qualche suo assegnamento la minor forza, ma quella che abbia un maggior scostamento fra la forza minore e la forza media relativa a tutti i suoi possibili assegnamenti. Tuttavia in 4.2.3.1 si è scelto di porre ad un valore arbitrariamente alto la forza relativa ad assegnamenti di operazioni non più possibili perchè violanti vincoli sulle risorse. In quel frangente non ci si è preoccupati di determinare con precisione questo valore arbitrario perchè era sufficiente che fosse abbastanza elevato da risultare maggiore di qualsiasi forza calcolabile in modo tradizionale. Questo introduce un problema però nel calcolo della forza media di un'operazione: come devono essere considerate le forze bloccate ai fini del calcolo della media delle forze? Infatti poichè il valore di una forza bloccata è arbitrario (si impone almeno che tutte le forze bloccate abbiano il medesimo valore), valori diversi potrebbero cambiare il candidato al successivo assegnamento. Esistono diverse soluzioni a questo problema. Quale sia meglio utilizzare non è immediato e potrebbe essere deciso sulla base di una serie di test approfonditi. Le possibilità per calcolare le forze sono:

- considerare le forze bloccate come normali forze e quindi utilizzare il valore arbitrario nel calcolo della media ipotizzando che le variazioni dovute alla variazione di tale valore non siano significative;
- fare la media delle forze non bloccate e trascurare quelle bloccate;
- considerare le forze bloccate come aventi un valore pari a quello peggiore fra le forze non bloccate;
- considerare le forze bloccate come aventi valore nullo; questo provoca rispetto al secondo caso un aumento dello scostamento forza media-forza migliore nel caso di forza media negativa e una sua diminuzione nel caso opposto.

4.2.5 Riflessioni sulla priorità delle operazioni per questa versione dell'algoritmo

In 4.1.1 è stato introdotto il concetto di priorità delle diverse operazioni e si è suggerito di introdurre nell'algoritmo la schedulazione delle operazioni in base alla loro appartenenza ad un blocco basico piuttosto che un altro. Non è stato tuttavia fornito un criterio su come ordinare i blocchi basici e quindi su come ordinare i diversi gruppi di operazioni. Sulla base delle considerazioni fatte nelle sezioni precedenti è ora possibile fornire un nuovo criterio di ordinamento, seppure anch'esso parziale.

Si considerino tre blocchi basici: un blocco coincidente con il ramo then di un costrutto condizionale (blocco *then*), un blocco coincidente con il ramo else dello stesso costrutto condizionale (blocco *else*) e il blocco successivo al termine della struttura condizionale e quindi successivo ad entrambi i rami che verrà chiamato blocco *endif*. Si consideri che non vi siano dipendenze fra le operazioni del terzo blocco e le altre: in questo caso le finestre temporali delle operazioni dei tre blocchi molto probabilmente si sovrapporranno per più di passi di controllo. Si avranno quindi operazioni dei tre blocchi schedulabili nello stesso passo di controllo. L'ordine dato ai blocchi basici condizionerà come si è visto l'ordine in cui le operazioni verrano schedulate. Si ipotizzi che le forze inducano a

schedulare tre operazioni di uno stesso tipo nello stesso passo di controllo anche se ciò non fosse possibile a causa del ridotto numero di unità funzionali e che le soluzioni ammissibili prevedano che venga schedulata in quel passo di controllo solo l'operazione non condizionata cioè appartenente al blocco *endif*. Se venissero assegnate prima le operazioni dei blocchi basici appartenenti ai rami condizionali, non essendo esse in conflitto per l'uso di risorse non violerebbero alcun vincolo. Solo al momento di schedulare le operazioni appartenenti al terzo blocco basico, l'algoritmo calcolerebbe le forze della terza operazione e si accorgerebbe che non ne è più possibile lo scheduling. Se invece il blocco *endif* venisse schedulato per primo, al momento del calcolo delle forze del secondo blocco, sia esso quello *then* o quello *else* l'algoritmo si accorgerebbe dell'errore.

In generale si può dire che l'algoritmo debba cercare di far incastrare i pezzi di SDG relativi ai diversi blocchi basici: quelli che non sono figli di nessun costrutto condizionale non saranno in mutua esclusione con nessun altro blocco, quindi c'è il concreto rischio che se venissero considerati per ultimi sarebbe difficile riuscirli ad incastrarli con tutti, cioè c'è il rischio che le operazioni ad essi appartenenti non trovino più risorse libere per essere eseguite perchè già completamente occupate. Conviene quindi che essi vengano schedulati immediatamente in modo tale da considerare i conflitti con gli altri blocchi basici, che sicuramente esistono, mano mano che questi ultimi verranno schedulati. Estendendo le considerazioni al caso di più costrutti condizionali annidati, si può dedurre che convenga lasciare per ultima la schedulazione dei blocchi basici aventi conflitto per l'accesso con le risorse con il minor numero possibile di blocchi: le operazioni di tali blocchi saranno in conflitto con un numero minore di operazioni rispetto a quello degli altri blocchi.

Il criterio quindi che verrà utilizzato per ordinare i blocchi basici cioè i gruppi di operazioni da schedulare è il numero decrescente di blocchi con cui essi sono in mutua esclusione. Tale ordine è ancora una volta però parziale perchè non vengono forniti metodi per ordinare i blocchi a parità di mutue esclusioni, ma verrà utilizzato un sottoordinamento casuale.

Questo criterio in realtà non produce significativi miglioramenti nei risultati, perchè cambia semplicemente l'ordine con cui considerare le operazioni. Tuttavia vi è un miglioramento significativo nella tempestività con cui vengono

portati allo scoperto le violazioni ai vincoli sulle risorse che anche se non manifeste possono essere già ormai inevitabili in uno scheduling parziale. Esistono infatti degli scheduling parziali che seppure non violano alcun vincolo non possono essere contenuti in alcuna soluzione accettabile. Scoprire tempestivamente questa caratteristica in uno scheduling consente di evitare parecchi passi di backtracking e pertanto di ridurre sensibilmente il tempo di computazione dell'algoritmo.

4.2.6 Analisi della complessità dell'Algoritmo

Nel caso in cui durante la costruzione dello scheduling l'algoritmo non violi nessun vincolo tecnologico, la sua complessità rimane $O(\frac{c^2n^3}{B})$ (c è il numero di passi di controllo, n il numero delle operazioni, B il numero di Blocchi Basici, f il numero di tipi di unità funzionali) in quanto l'overhead per la gestione delle strutture dati necessarie a gestire la possibilità del backtracking è trascurabile. Rimane da considerare cosa succede se l'algoritmo deve eseguire qualche passo di backtracking. Seguendo la teoria dei grafi e considerando il caso pessimo la complessità risulta essere addirittura fattoriale, in quanto il grado dell'albero dipende dalla dimensione del problema, e pari a $O(\frac{ncf}{B}^{\frac{ncf}{B}})$. Introducendo la limitazione suggerita in 4.2.3.5 (Grado dell'albero di Ricerca) la complessità del caso pessimo teoricamente rimane uguale perchè si deve considerare la possibilità che si trovi una soluzione accettabile solo all'ultima espansione del grado dell'albero. In realtà però è molto difficile trovare problemi di scheduling in cui non si scopra una soluzione accettabile ponendo il grado dell'albero di ricerca pari a 4. Pertanto la vera complessità temporale del caso pessimo si può considerare esponenziale e pari a $O(4^n)$. Questa rimane comunque una considerazione sul caso pessimo: la complessità del caso medio può essere modellizzata come $O((n+p)\frac{c^2n^3}{B})$ dove p indica il numero di backtracking che si sono resi necessari. Questo valore ovviamente è possibile stabilirlo sono al termine dell'esecuzione dell'algoritmo perchè dipende non solo dalla dimensione del problema ma anche dal valore dei dati in ingresso. Mediamente tuttavia il numero di passi di backtracking è nello stesso ordine di grandezza del numero di operazioni. In conclusione quindi possiamo definire la complessità media del-

l'algoritmo modificato come $O(\frac{ic^2n^3}{B})$ dove i è un intero positivo che dipende dalle caratteristiche dal particolare problema di scheduling da risolvere.

Capitolo 5

L'Implementazione all'interno del Progetto PandA

L'algoritmo del Force Directed scheduling modificato come indicato nel capitolo 4 è stato implementato sia per verificare il guadagno reale in termini di qualità del risultato e tempo di computazione grazie a quanto mostrato in 4.1 (Proposte di modifiche all'algoritmo originale), sia per verificare il corretto funzionamento e i risultati ottenibili nel caso di introduzione di vincoli sul numero delle risorse disponibili (4.2). L'implementazione è stata fatta all'interno del progetto PandA ([mlDdEeIPdM06]) per sfruttare i servizi e le funzionalità da esso forniti. Questo progetto verrà descritto nella prima parte di questo capitolo, mentre la seconda parte fornirà alcune informazioni relative all'implementazione realizzata dell'algoritmo di scheduling.

5.1 PandA Project

PandA è un progetto sviluppato dal Laboratorio di Microarchitetture del Dipartimento di Elettronica e Informazione presso il Politecnico di Milano: l'obiettivo primario del progetto è la realizzazione di un framework in ambiente Linux per la ricerca e la sperimentazione nel campo dell'HW-SW Co-Design. Uno dei suoi sottoprogetti principali all'interno del quale il lavoro presentato

si inserisce è quello riguardante la realizzazione di uno strumento automatico per la sintesi ad alto livello. Gli ingressi di questo strumento sono descrizioni di sistema scritte in C, C++ o SystemC mentre il risultato prodotto è costituito da descrizioni in VHDL a livello RTL sintetizzabile.

PandA è realizzato modularmente per permetterne facilmente l'espansione e l'integrazione con nuove funzionalità e nuovi algoritmi. Di seguito verrà data una breve descrizione dei moduli o sottoprogetti con i quali l'algoritmo presentato in questo lavoro di tesi interagisce.

5.1.1 Analizzatore delle specifiche

Scopo di questo sottoprogetto è trasformare la descrizione di sistema scritta in C, C++ o SystemC in una struttura ad albero. Per facilitare quest'attività si utilizza una delle possibilità offerte dal compilatore GNU GCC ([Fou06]): scrivere su file la struttura dell'albero di sintassi che rappresenta il codice sorgente compilato. In questo modo sono ridotte le funzionalità che devono essere implementate a partire da zero direttamente in questo sottoprogetto, fra le quali la parte di ottimizzazione del codice. Il file prodotto viene poi letto dall'analizzatore per estrarre da esso le informazioni necessarie e per identificare le estensioni SystemC presenti. Questi dati vengono memorizzati in una struttura dati anch'essa ad albero che è passata al successivo modulo.

5.1.2 Costruttore dei grafi

A partire dalla struttura ad albero prodotta dal sottoprogetto precedente vengono costruiti una serie di grafi utilizzando le librerie BOOST ([boo06]). Tali grafi servono a mettere più chiaramente in mostra i diversi tipi di dipendenze presenti fra le varie coppie di operazioni e vengono successivamente utilizzati da diversi moduli di PandA, come ad esempio gli schedulatori. Fra i molteplici grafi prodotti da questo modulo va sottolineata la presenza di quelli che saranno poi utilizzati dal Force Directed scheduling: l'SDG, il grafo dei Blocchi Basici e l'Albero dei Dominatori.

PandA offre anche, soprattutto per funzioni di debugging, la possibilità di costruire questi grafi a partire da descrizione testuali scritte in pseudo-c.

5.1.3 Lettore delle informazioni sulla tecnologia e sui vincoli

PandA prevede che attraverso due ulteriori file vengano forniti al motore di sintesi ad alto livello sia alcune delle caratteristiche della tecnologia su cui si vuole mappare la funzionalità, sia i vincoli da imporre. In particolare con caratteristiche della tecnologia si intende il fornire per ciascuna delle unità funzionali allocabili:

- il nome dell'unità funzionale;
- le operazioni da essa eseguibili;
- il tempo di esecuzione di ciascuna operazione;
- il numero di porte di ingresso e di uscita e il tipo di dati;
- un indice dell'area occupata.

I vincoli forniti riguardano invece la durata del periodo di clock e il numero massimo di unità allocabili per ogni tipo. E' possibile non fornire alcun dato riguardo alla tecnologia: in tal caso PandA utilizzerà quella di default memorizzata all'interno del framework. Se non si impongono vincoli relativamente al numero di unità funzionali di un certo tipo, PandA presupporrà che esse siano potenzialmente allocabili in numero infinito.

5.1.4 Schedulatori

Il lavoro di tesi presentato fa parte di questo sottoprogetto e contemporaneamente ne usa dei componenti. Da un punto di vista del codice sorgente, come verrà evidenziato in 5.2 la classe che implementa il Force Directed scheduling eredita dalla classe `scheduling` progenitrice di tutte le classi che implementano algoritmi di scheduling; da un punto di visto logico il Force Directed scheduling

utilizza direttamente i dati calcolati da ASAP e ALAP. L'ALAP stesso utilizza dei dati forniti dal List-Based, in particolare il numero di passi di controllo totale, quindi l'algoritmo qui presentato utilizza, anche se indirettamente, il List-Based.

5.1.5 Sintetizzatore dell'unità di controllo

I risultati prodotti dagli algoritmi implementati nel sottoprogetto appena descritto vengono utilizzati dal sintetizzatore dell'unità di controllo. In particolare l'unità di controllo viene modellizzata seguendo il paradigma delle FSMD ([GR94], [FS] e [Bra05]) e quindi come macchina a stati finiti. Il controllore deve conoscere quali operazioni il datapath deve eseguire ad ogni passo di controllo per poter pilotare correttamente attraverso segnali l'attivazione delle unità funzionali, la scrittura dei dati nei registri, l'eventuale generazione di indirizzi per il trasferimento di dati e i segnali di controllo per i multiplexer. Quali operazioni debbano essere eseguite nei diversi passi di controllo costituisce proprio l'informazione generata dagli algoritmi di scheduling.

5.2 Implementazione del Force Directed Scheduling

L'implementazione del Force Directed scheduling legge i dati del problema che sono stati ricavati da altri moduli di PandA, in particolare da istanze delle classi `graph_manager` e `technology_manager`. Una volta computata la soluzione al problema presentato, i dati ad essa relativa vengono restituiti attraverso una struttura di tipo `HLS` in cui vengono memorizzati a quale passo di controllo e a quale tipo di unità funzionale sono state assegnate le singole operazioni. Oltre alla classe `force_directed` sono state create altre classi ausiliari: `stackSchedule`, `FDschedule`, `infOp_type` e `job`.

5.2.1 Scelte implementative

Nel capitolo 4 è stata fornita la descrizione di una nuova versione del Force Directed scheduling. Tale descrizione ha evidenziato come a differenza della

versione originale quella proposta presenti dei parametri il cui valore può modificare i risultati e il tempo di esecuzione dell'algoritmo. Per poter determinare il valore ottimo di questi parametri sarebbe stata necessaria l'applicazione di ulteriori algoritmi (per esempio algoritmi genetici o a raffinamenti successivi) o l'applicazione di sessioni di test esaustive sufficienti ad offrire una buona copertura di tutto lo spazio dei possibili problemi di scheduling. Quest'attività molto complessa, soprattutto per l'individuazione del set di test adeguato, non fa parte di questo lavoro in quanto comunque si ritiene che utilizzare parametri ottimali rispetto a quelli ipotizzati non produca un evidente aumento nella qualità dei risultati o nei tempi di esecuzione dell'algoritmo.

Pertanto i parametri sono stati fissati, in parte arbitrariamente ed in parte sulla base dell'esperienza maturata nei test, in questo modo:

- grado iniziale dell'albero: 2
- variazione del grado dell'albero al termine della sua esplorazione infruttuosa: il nuovo grado è pari al doppio del precedente grado
- peso applicato alla forza relativa ad un predecessore o successore negativa: 0.001
- valore fissato di una forza relativa ad un assegnamento impossibile o che provoca un assegnamento impossibile: 100
- scelta nella funzione per stabilire il prossimo assegnamento da effettuare (confrontare 4.2.4): è stata introdotta la possibilità di lasciare la scelta all'utente; in questo modo è possibile confrontare i risultati su qualsiasi problema.

5.2.2 Diagrammi di classe e Diagrammi di Collaborazione

Vengono ora elencate le classi realizzate per implementare il Force Directed scheduling e riportati i loro Diagrammi di Classe/di Collaborazione:

5.2.2.1 stackSchedule

Un oggetto di questa classe memorizza i dati di un assegnamento effettuato o di un assegnamento bloccato in quanto già effettuato e successivamente cancellato. All'interno di esso sono memorizzati:

- i dati relativi all'operazione: identificatore, blocco basico di appartenenza, priorità;
- i dati relativi all'assegnamento: tipo di unità funzionale e passo di controllo;
- se l'assegnamento è stato effettuato o bloccato;
- l'ASAP e l'ALAP dell'operazione precedenti all'assegnamento; questi dati sono utilizzati quando e se l'assegnamento viene annullato;
- la finestra temporale influenzata da questo assegnamento, cioè i passi di controllo in cui la percentuale di occupazione riferita a qualche tipo di unità funzionale è stata modificata a seguito dell'assegnamento stesso; anche questi dati vengono utilizzati nel momento dell'eventuale annullamento dell'assegnamento.

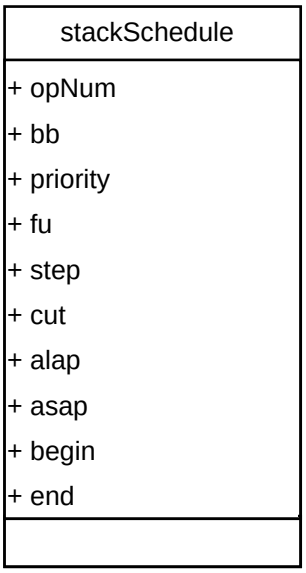


Figura 5.1. Diagramma della classe stackSchedule

5.2.2.2 FDschedule

Un oggetto di questa classe memorizza i dati di un possibile assegnamento non ancora effettuato. All'interno di esso sono memorizzati:

- i dati relativi all'operazione: identificatore, blocco basico di appartenenza, priorità;
- i dati relativi all'assegnamento: tipo di unità funzionale, passo di controllo e se è bloccato o meno;
- il valore della forza corrispondente;
- la finestra temporale influenzata da questo assegnamento, cioè i passi di controllo in cui la percentuale di occupazione riferita a qualche tipo di unità funzionale è stata modificata a seguito dell'assegnamento stesso; questi passi di controllo coincidono con quelli in cui la variazione della percentuale di occupazione relativa ad un tipo di unità funzionale potrebbe provocare la variazione di forza relativa all'assegnamento contenuto nell'istanza; questi dati sono utilizzati dall'algoritmo per evitare di

ricalcolare ad ogni iterazione tutte le forze e limitarsi al calcolo di quelle modificate dall'ultimo assegnamento o dall'ultima cancellazione di un assegnamento.

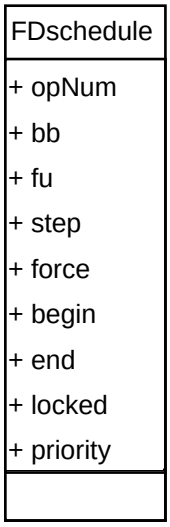


Figura 5.2. Diagramma della classe FDschedule

5.2.2.3 infOp_type

Un oggetto di questa classe memorizza i dati relativi alla percentuale di occupazione di un certo tipo di unità funzionale da parte di un'operazione in un passo di controllo. Da questi dati si ricavano le percentuali di occupazione complessive. La coppia <unità funzionale-passo di controllo> non è indicata al suo interno perchè la classe `force_directed` costruisce una lista di `infOp_type` diversa per ciascuna coppia. All'interno di un'istanza di `infOp_type` vengono memorizzati:

- i dati dell'operazione: identificatore, blocco basico di appartenenza e se è gia stata assegnata;
- la percentuale di occupazione;
- l'elenco dei blocchi basici in mutua esclusione con quello dell'operazione indicata nell'istanza stessa e ai quali appartiene almeno un'operazione

schedulabile con probabilità non nulla nello stesso passo di controllo e sullo stesso tipo di unità funzionale del possibile assegnamento sottinteso dall'istanza stessa; queste liste sono utilizzate per limitare il numero di blocchi basici da collassare durante il calcolo delle percentuali di occupazione nel caso di presenza di costrutti condizionali (5.2.3);

- nel caso di operazioni multiciclo se l'operazione può essere schedulata nel passo di controllo sottinteso o se l'occupazione dell'unità funzionale si riferisce ad una operazione schedulabile nei passi di controllo precedenti la cui esecuzione potrebbe non essere ancora terminata nel passo di controllo sottinteso.

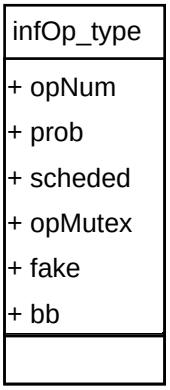


Figura 5.3. Diagramma della classe infOp_type

5.2.2.4 job

Un oggetto di questa classe memorizza i dati relativi alla restrizione di un predecessore o di un successore a seguito di uno scheduling. Ogni volta che si calcola una forza viene creata una lista di questi oggetti per memorizzare la restrizione della finestra di mobilità di ogni predecessore o successore; per ciascuna restrizione si dovrà calcolare il contributo alla *predecessors' and successors' force* corrispondente.

Dopo aver calcolato il contributo relativo alla restrizione della finestra di un predecessore (successore), si calcolano anche le eventuali riduzioni di mobilità cui sono soggetti i suoi diretti predecessori (successori) e si aggiungono

alla lista. Mantenendo gli oggetti ordinati in base all'ordinamento topografico delle operazioni nel SDG (questo se si riferiscono a restrizioni delle mobilità dei successori; per i predecessori l'ordine è quello inverso) si garantisce che il contributo di forza relativo ad un predecessore o ad un successore verrà calcolato un'unica volta. Infatti se si fossero usate tecniche ricorsive per il calcolo delle forze poteva accadere per esempio che il contributo di un predecessore, se questo era legato tramite due diversi percorsi all'operazione di cui si sta calcolando la forza, venisse considerato due volte. Le informazioni memorizzate da un oggetto di questa classe sono:

- identificatore dell'operazione;
- posizione dell'operazione nell'ordinamento topologico del SDG;
- nuova finestra temporale dell'operazione a seguito dell'ipotesi di assegnamento;
- mobilità dell'operazione a seguito dell'ipotesi di assegnamento (esplicitata per permettere l'ordinamento fra due istanze relative alla stessa operazione).

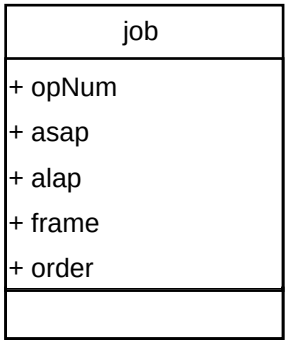


Figura 5.4. Diagramma della classe job

5.2.2.5 force_directed

La classe `force_directed` implementa il Force Directed scheduling; eredita come tutti gli algoritmi di scheduling implementati in PandA dalla classe `scheduling`. Le strutture dati contenute in essa memorizzano:

- i dati del problema di scheduling:
  - il grafo SDG;
  - le caratteristiche delle unità funzionali e i vincoli sul loro numero;
  - l'ordinamento topologico, il tipo, l'ASAP e l'ALAP delle operazioni;
  - le caratteristiche dei diversi tipi di operazione;
- la priorità assegnata alle operazioni;
- il grado attuale dell'albero di ricerca;
- il criterio di scelta del prossimo assegnamento selezionato;
- le operazioni già assegnate e il numero di operazioni ancora da assegnare;
- il numero di soluzioni anche parziali che non soddisfano i vincoli incontrate durante la ricerca;
- l'elenco delle mutue esclusioni fra blocchi basici e fra operazioni;
- per ciascun tipo di unità funzionale il numero utilizzato in ciascun passo di controllo e il binding delle operazioni sulle unità funzionali;
- le forze relative a tutti gli assegnamenti possibili;
- le percentuali di occupazione, totali e relative alle singole operazioni, dei diversi tipi di unità funzionale nei diversi passi di controllo.

La classe oltre al costruttore fornisce due metodi pubblici: il primo computa l'algoritmo, il secondo stampa informazioni riguardo ad esso. I metodi privati invece sono più numerosi; i servizi che essi offrono sono:

- inizializzazione delle strutture dati riguardanti i dati del problema di scheduling;
- inizializzazione delle strutture dati riguardanti le unità funzionali;
- eliminazione di un'operazione dalle liste contenenti le percentuali di occupazione relative;

- inserimento di un'operazione nelle liste contenenti le percentuali di occupazione relative;
- calcolo delle mutue esclusioni fra operazioni schedulabili in un certo passo di controllo;
- calcolo delle percentuali di occupazione totali;
- restrizione della finestra temporale di un'operazione e conseguente ricalcolo delle percentuali di occupazione;
- calcolo degli immediati predecessori di un'operazione;
- calcolo degli immediati successori di un'operazione;
- calcolo della forza di un assegnamento;
- calcolo della *self-force* di un assegnamento;
- calcolo delle *predecessors' and successors' force* di un assegnamento;
- calcolo del prossimo assegnamento da effettuare;
- calcolo del miglior tempo di esecuzione di un'operazione;
- calcolo del binding di un'operazione su un'unità funzionale;
- cancellazione di un binding già effettuato.

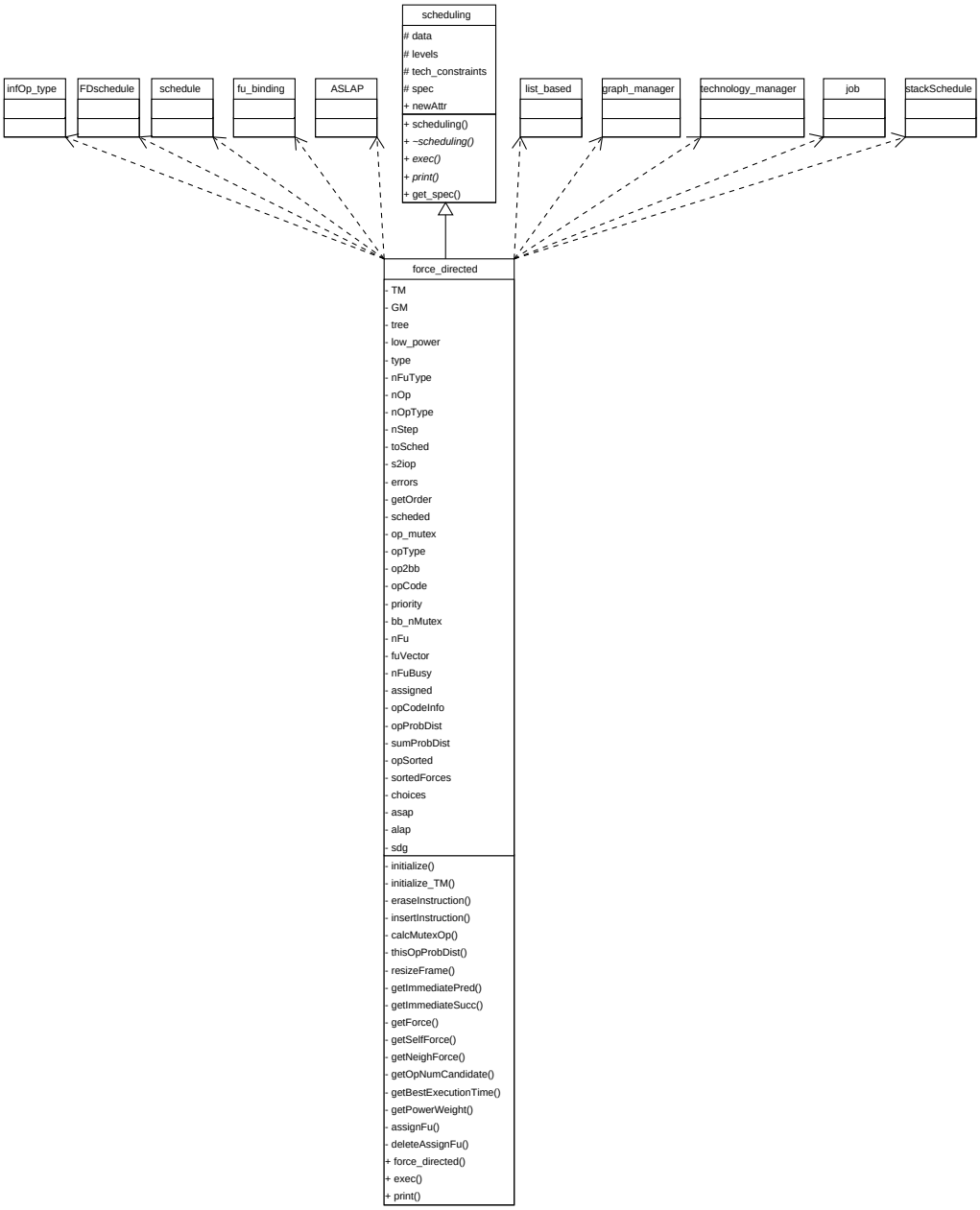


Figura 5.5. Diagramma della classe force_directed

5.2.3 Calcolo delle somme di probabilità

Viene ora fornita la soluzione adottata nell'implementazione per risolvere il problema di calcolare le somme di probabilità (e quello equivalente di calcolare le percentuali di occupazione) indicate in 3.1.1. Questo sia perchè da un punto di vista implementativo questa funzione risulta la più critica, sia perchè in letteratura non si trovano descrizioni di possibili metodi per risolvere questo particolare problema ed esso non è di facile soluzione se non con metodi esaustivi e quindi di complessità esponenziale.

La funzione lavora su liste di operazioni che possono essere eseguite da un tipo di unità funzionale in un determinato passo di controllo. Tali operazioni sono ordinate secondo due chiavi:

1. il numero di blocchi basici che sono in mutua esclusione con quello dell'operazione e a cui appartiene almeno un'altra operazione della lista;

2. l'ordinamento lessicografico (secondo la rappresentazione interna) del blocco basico cui appartengono.

La prima chiave, per la quale si usa l'ordinamento decrescente (l'ordinamento utilizzato per la seconda chiave è ininfluente così come l'ordinamento fra operazioni che abbiano le stesse chiavi), che è facilmente calcolabile, comporta che i blocchi basici che costituiscono i diversi rami dei costrutti condizionali annidati più internamente si trovino nelle prime posizioni della lista, mentre i blocchi basici relativi a segmenti non condizionati del problema si trovano nella sua parte terminale.

Per ogni blocco basico, una cui operazione è presente nella lista, si crea una struttura dati che contiene le informazioni relative al blocco basico stesso, le due chiavi utilizzate per l'ordinamento delle operazioni, una lista di quali siano i blocchi basici con cui esso è in mutua esclusione e un valore detto probabilità pari alla somma delle probabilità delle operazioni che si trovano nella lista e che appartengono a quel blocco basico. I blocchi basici che hanno la prima chiave pari a zero vengono trascurati e la probabilità delle operazioni ad essi appartenenti viene sommata direttamente alla soluzione finale che inizialmente è posta

pari a zero. Gli altri blocchi basici vengono ordinati in una nuova lista con lo stesso tipo di ordinamento illustrato per le operazioni.

A questo punto scorrendo la lista si individua la coppia di blocchi basici appartenenti ai rami di uno dei costrutti condizionali più annidati e si collassano: le informazioni del nuovo blocco basico coincideranno con quelle di uno qualsiasi dei due ad eccezione della probabilità che sarà pari al valore maggiore fra le due. Il blocco basico così creato potrebbe trovarsi concatenato con uno o più blocchi basici (concatenazione in questo senso: due o più blocchi basici sono concatenati se sono legati da archi di controllo e nessun altro blocco basico ad eccezione dei predecessori del primo e dei successori dell'ultimo è legato con un arco di controllo ad uno di essi). I blocchi concatenati si fondono mantenendo le informazioni di uno qualsiasi di essi ad eccezione della probabilità che viene posta pari alla sommatoria di quelle dei blocchi originari. In questo modo è come se nel grafo dei blocchi basici fosse stata eliminata la biforcazione relativa al costrutto condizionale più annidato. Ad ogni iterazione si mantiene ordinata la lista dei blocchi basici che vede ridursi il numero dei suoi elementi. A questo punto basta applicare ripetutamente quanto descritto fino ad eliminare la biforcazione nel grafo relativa all'ultimo costrutto condizionale superstite. La probabilità del blocco superstite verrà sommata a quella della soluzione ottenuta fino a quel momento ottenendo così il valore definitivo.

La complessità di questo metodo è pari a $O(B^4)$ dove B è il numero dei blocchi basici presenti nel grafo omonimo.

5.2.4 Calcolo della mutua esclusione fra le operazioni

Un altro aspetto critico nell'implementazione dell'algoritmo è la parte che riguarda il calcolo delle mutue esclusioni fra operazioni. La soluzione proposta lavora a livello di blocchi basici (se due operazioni appartengono a due blocchi basici in mutua esclusione allora saranno esse stesse in mutua esclusione) e si basa sull'Albero dei Dominatori.

Inizialmente si compie una visita inversa breadth-first dell'Albero dei Dominatori durante la quale si memorizzano i discendenti di ciascun nodo. A questo punto si esegue nuovamente una visita breadth-first (questa volta non inversa)

dell'Albero dei Dominatori. Gli assegnamenti da compiere durante la seconda visita si differenziano a seconda del tipo di blocco basico:

- Nodo Radice:

  nessun assegnamento

- Nodo di un blocco basico corrispondente ad un ramo then:

  la lista dei blocchi basici con cui esso è in esclusione è posta uguale all'unione di:

  - il blocco basico che corrisponde ad un ramo else e il cui nodo è fratello di quello del nodo in oggetto (se presente);
  - i discendenti del blocco basico indicato al punto precedente (se presente) che sono già stati calcolati durante la prima visita;
  - i blocchi basici in mutua esclusione con il blocco basico rappresentato dal genitore del nodo in oggetto; sono già stati calcolati perchè la seconda visita è discendente.

- Nodo di un blocco basico corrispondente ad un ramo else:

  la lista dei blocchi basici con cui esso è in esclusione è posta uguale all'unione di:

  - il blocco basico che corrisponde ad un ramo then e il cui nodo è fratello di quello del nodo in oggetto;
  - i discendenti del blocco basico indicato al punto precedente che sono già stati calcolati durante la prima visita;
  - i blocchi basici in mutua esclusione con il blocco basico rappresentato dal genitore del nodo in oggetto; sono già stati calcolati perchè la seconda visita è discendente.

- Nodo di un blocco basico di tipo endif:

  la lista dei blocchi basici con cui esso è in esclusione è posta uguale alla lista delle mutue esclusioni del padre già calcolata.

Capitolo 6

Risultati Sperimentali

In questo capitolo vengono presentati i risultati sperimentali ottenuti utilizzando l'algoritmo del Force Directed scheduling modificato proposto nel capitolo 4 confrontati con quelli della versione originale dell'algoritmo e con quelli forniti dal List-Based. I test sono stati fatti utilizzando l'implementazione realizzata all'interno di PandA. I benchmark utilizzati sono quelli forniti con PandA stesso, in particolare quelli forniti sotto forma di moduli scritti in codice c. La macchina utilizzata per i test è un portatile basato su architettura Intel Sonoma con processore a 1,83 Ghz e 1 GB di memoria RAM PC4300.

In particolare i risultati analizzati sono quelli riferiti a:

- tempo di esecuzione;
- numero totale di unità funzionali allocate; il numero fornito è quello totale senza discriminazione fra i diversi tipi di unità funzionale perchè non sono stati inseriti nei dati relativi all'architettura i costi delle diverse unità funzionali, quindi per l'algoritmo minimizzare l'utilizzo di un particolare tipo di un'unità funzionale piuttosto che un altro è irrilevante. Unica eccezione sono i casi caratterizzati dalla presenza di vincoli in cui l'algoritmo distingue fra unità vincolate e unità non vincolate (6.6).

In alcuni casi al posto del tempo di esecuzione è riportato il termine MAX. Questo sta a significare che quella computazione dell'algoritmo ha richiesto più

di un migliaio di secondi. In tal caso si è ritenuto che il tempo di esecuzione fosse comunque troppo elevato e quindi non significativo. Un tempo di esecuzione così elevato può manifestarsi solo nel caso di un elevato numero di backtracking e quindi in caso di presenza di vincoli sulle risorse. In particolare si è riscontrato questo problema nell'esecuzione della versione pseudo-originale del Force Directed scheduling (per versione pseudo-originale si intende la versione originale cui sono state fatte leggere modifiche per permettere di aggiungere il backtracking e consentire così di eliminare le soluzioni non accettabili).

Per il numero relativamente elevato delle modifiche introdotte sono state effettuate sessioni di test diverse per mostrare gli incrementi relativi a singole variazioni o a gruppi di esse.

Nelle tabelle con *nFU* si indica il numero totale di unità funzionali allocate dall'algoritmo, con *tempo* il tempo necessario per la computazione dello scheduling in secondi.

6.1 Dimensioni dei benchmark

Si riportano numero di operazioni e numero di salti condizionati presenti nei benchmark utilizzati per testare le diverse versioni dell'algoritmo:

Benchmark	nOp	n° salti
Adpcm_decode	68	11
Adpcm_encode	83	14
Arf	34	0
Bandpass	52	0
Chemical	38	0
ComputeForwVector	77	14
Dct	58	0
Dct_wang	57	0
Diffeq	27	1
Ewf	39	0
Ewf_v2	55	0
Kim	34	2
Maha	30	5

Tabella 6.1. Numero di operazioni e di salti condizionati di ogni benchmark

6.2 Variazioni nel calcolo della forza

Vengono qui proposti i risultati sperimentali che mostrano i guadagni nella qualità dei risultati ottenuti tramite la modifica della formula per il calcolo della forza introducendo la variazione della somma di probabilità al posto della variazione della probabilità della singola operazione (4.1) e limitando il contributo dovuto a *predecessors' and successors' forces* negative (4.1.3).

In particolare per i test in tabella 6.2, in cui non sono presenti costrutti condizionali, l'unica variazione effettiva è quella prodotta dalla variazione nel calcolo delle *predecessors' and successors' forces*.

I test di tabella 6.3 presentando anche costrutti condizionali mostrano invece anche i miglioramenti dovuti all'introduzione della variazione della somma di probabilità.

Benchmark	FD Orig.		FD Modif.	
	nFU	tempo	nFU	tempo
Arf	14	0.23	8	0.25
Bandpass	20	0.55	11	0.57
Chemical	25	0.45	15	0.69
Dct	35	0.80	20	1.03
Dct_wang	29	0.65	20	0.85

Tabella 6.2. Tabella comparativa calcolo Forza originale - calcolo Forza modificato nei test senza costrutti condizionali

Benchmark	FD Orig.		FD Modif.	
	nFU	tempo	nFU	tempo
Adpcm_decode	27	2.89	18	10.69
Adpcm_encode	27	5.84	19	43.68
Diffeq	14	0.09	8	0.29
Kim	9	0.46	6	0.73
Maha	10	0.39	5	0.43

Tabella 6.3. Tabella comparativa calcolo Forza originale - calcolo Forza modificato nei test con costrutti condizionali

Apparentemente il guadagno in termini di unità funzionali risparmiate è notevole, tale da giustificare largamente l'aumento del tempo di esecuzione. Tale aumento è dovuto in primo luogo alla maggiore complessità introdotta nel calcolo della forza dovuta al calcolo della differenza della somma di probabilità e in secondo luogo al fatto che nella versione originale dell'algoritmo la mobilità delle operazioni si riduce molto più velocemente, quindi diminuisce più velocemente il numero di forze da calcolare. Il guadagno è in realtà meno sostanziale di quanto appaia perchè la diminuzione riguarda anche, ma non solo, l'utilizzo di pseudo-unità funzionali che vengono introdotte per eseguire pseudo-operazioni aggiunte dal gcc nella sua rappresentazione interna. Di fatto in alcuni casi l'area occupata da queste pseudo-unità è nulla. Il guadagno in termini di costo d'area nei test proposti è quindi più limitato di quanto appaia inizialmente, ma comunque significativo.

6.3 Introduzione della priorità delle operazioni

Vengono qui proposti i risultati sperimentali che mostrano i guadagni dovuti all'introduzione della priorità delle operazioni mostrata in 4.1.1. Ovviamente i test comparativi sono stati effettuati solo su benchmark che presentano costrutti condizionali in quanto in assenza di questi i risultati restano identici (a meno di una leggera maggiorazione nei tempi di computazione dovuta alla gestione delle strutture dati necessarie a memorizzare la priorità delle operazioni). In entrambe le versioni sono state applicate le modifiche suggerite in 4.1 e 4.1.3 (correzione nel calcolo della forza).

Benchmark	FD Orig.		FD Modif.	
	nFU	tempo	nFU	tempo
Adpcm_decode	18	11.77	18	3.54
Adpcm_encode	19	43.30	19	7.92
Diffeq	8	0.28	8	0.25
Kim	6	0.68	6	0.58
Maha	5	0.41	5	0.25

Tabella 6.4. Tabella comparativa Force Directed senza priorità - Force Directed con Priorità

Come è facile osservare il guadagno in termini di velocità nella computazione senza perdite di qualità di risultati è significativo soprattutto per quei problemi con numero elevato di operazioni e numero medio di operazioni per blocco basico basso.

6.4 Cambio del criterio di scelta del prossimo assegnamento

I risultati dei test effettuati hanno evidenziato che il miglior criterio di scelta del prossimo assegnamento fra quelli proposti in 4.2.4 è l'ultimo di quelli elencati (questo in particolare riferendosi ai test mostrati in 6.4.2; per quanto riguarda i test proposti in 6.4.1 i diversi criteri di scelta del prossimo assegnamento danno risultati identici in quanto non esistendo vincoli non esistono neppure forze

bloccate), cioè considerare pari a zero al fine del calcolo della forza media le forze bloccate. Si mostrano i risultati relativi a due classi diverse di problemi:

6.4.1 Test senza vincoli sulle risorse

Benchmark	FD Orig.		FD Modif.	
	nFU	tempo	nFU	tempo
Adpcm_decode	18	3.49	18	4.33
Adpcm_encode	19	7.93	20	8.04
Arf	8	0.21	8	0.21
Bandpass	11	0.60	11	0.57
Chemical	15	0.64	15	0.68
Dct	20	0.94	20	0.82
Dct_wang	20	0.81	20	0.78
Diffeq	8	0.24	9	0.22
Kim	6	0.56	5	0.58
Maha	5	0.25	5	0.25

Tabella 6.5: Tabella comparativa criterio di scelta del prossimo assegnamento - casi senza vincoli

I risultati riportati in tabella 6.5 mostrano che in realtà il cambiare il criterio di scelta del prossimo assegnamento non comporta grandi differenze nei risultati sia dal punto di vista del numero di unità funzionali allocate, sia dal punto di vista del tempo di computazione. Tuttavia la versione finale dell'algoritmo proposta offre un criterio di assegnamento diverso da quello originario, nonostante per questa classe di problemi non comporti un effettivo guadagno, in virtù dei risultati esposti nella successiva sottosezione.

6.4.2 Test con vincoli sulle risorse

Vengono anticipati i risultati di alcuni test in caso di presenza di vincoli sulle risorse per giustificare il cambio nel criterio di scelta del prossimo assegnamento. I risultati riportati dimostrano il vantaggio in termini di tempo di esecuzione prodotto dal cambiamento di criterio. I vincoli imposti consistono nel fissare il numero di unità funzionali a :

1 sommatore

1 sottrattore

1 moltiplicatore

1 comparatore

1 shifter

2 unità per accesso ad array

I tipi di unità non elencati sono considerati allocabili in numero infinito.

Benchmark	FD Orig.		FD Modif.	
	nFU	tempo	nFU	tempo
Arf	3	17.95	3	8.86
Bandpass	7	114.27	7	6.06
Chemical	5	35.41	6	21.86
Dct	6	90.03	5	103.58
Dct_wang		MAX	9	192.97
Ewf		MAX	4	32.54

Tabella 6.6. Tabella comparativa criterio di scelta del prossimo assegnamento - casi con vincoli

Da notare come la versione con criterio di scelta modificato trovi una soluzione in tempi ragionevoli anche nei casi in cui la versione che utilizza il criterio originale non riesca a trovarne una entro il tempo di 1000 secondi e che in generale utilizzi comunque meno tempo per la computazione. Il peggioramento nel numero di unità di esecuzione allocate che si è verificato in un caso di test è relativo ad un'unità in grado di eseguire le pseudo-operazioni e pertanto ha poca importanza per il costo in termini di area.

6.5 Scheduling con vincoli sulle risorse

Vengono ora proposti i risultati nel caso di vincoli con le risorse. Si confrontano tempo di esecuzione e numero di unità allocate di due versioni diverse dell'algoritmo. I vincoli sono quelli riportati in 6.4.2:

- 1° versione: Force Directed modificato con l'introduzione delle modifiche proposte in 4.1, 4.1.3 e 4.2.3 (correzione nel calcolo delle forze e introduzione del backtracking);
- 2° versione: Force Directed modificato con tutte le variazioni proposte in questo lavoro.

Benchmark	FD 1° Vers.		FD 2° Vers.	
	nFU	tempo	nFU	tempo
Adpcm_decode	15	19.06	14	7.36
Adpcm_encode	12	93.98	14	30.82
Arf	4	17.95	5	8.86
Bandpass	7	114.27	8	6.06
Chemical	6	35.41	7	21.86
ComputeForwVector		MAX	11	15.73
Dct	5	103.58	6	90.03
Dct_wang		MAX	9	192.97
Diffeq	7	1.06	8	0.94
Ewf		MAX	4	32.54
Kim	5	1.28	5	0.55
Maha	5	0.18	5	0.13

Tabella 6.7. Tabella comparativa nel caso di scheduling con vincoli sulle risorse

I dati dicono che la seconda versione dell'algoritmo fornisce risultati mediamente leggermente peggiori nei casi in cui la prima versione produca la soluzione in tempo ragionevole. Tuttavia la seconda versione a differenza della prima ha prodotto una soluzione accettabile in tempo minore ai 1000 secondi in ogni test e comunque sempre in un tempo decisamente minore rispetto alla prima.

6.6 Confronto con i risultati del List Based in caso di vincoli sulle risorse

In un buon numero di test il Force Directed scheduling ottiene gli stessi risultati in numero di unità funzionali allocate del List Based con un tempo di com-

putazione sensibilmente maggiore. Questo poichè in realtà le unità funzionali più significative nei benchmark considerati erano già state vincolate e questo limita di molto le dimensioni dello spazio progettuale. Esistono tuttavia dei test in cui il Force Directed ha ottenuto una soluzione con un minor numero di unità funzionali allocate.

I vincoli imposti per tali test sono:

2 sommatore

2 sottrattore

1 moltiplicatore

2 comparatori

1 shifter

2 unità per accesso ad array.

6.6.1 Diminuzione delle unità funzionali soggette a vincoli

In questa sezione sono riportati due benchmark per i quali il Force Directed scheduling ha ottenuto per alcuni tipi di risorsa l'allocazione di un numero di unità inferiore a quello imposto dai vincoli.

6.6.1.1 Benchmark Kim

Unità Funzionale	Vincolo	LB	FD
indirect_ref	inf	1	2
MINUS	2	2	1
PLUS	2	2	2
READ_COND	inf	1	1
Tempo computazione		0.11	0.18

Tabella 6.8. Unità allocate per il benchmark Kim

Il risultato è da considerarsi positivo in quanto essendo indirect_ref una pseudo-operazione, l'area occupata da un'unità funzionale che possa eseguirla è sensibilmente inferiore a quella occupata da un sottrattore.

6.6.1.2 Benchmark Maha

Unità Funzionale	Vincolo	LB	FD
CMP	2	2	1
indirect_ref	inf	1	1
MINUS	2	2	1
PLUS	2	1	1
READ_COND	inf	2	1
Tempo computazione		0.14	0.12

Tabella 6.9. Unità allocate per il benchmark Maha

6.6.2 Diminuzione delle unità funzionali non soggette a vincoli

In questa sezione sono riportati alcuni fra i benchmark per i quali il Force Directed scheduling ha ottenuto per alcuni tipi di risorse non vincolate l'allocazione di un numero di unità inferiore a quello determinato dal List-Based.

6.6.2.1 Benchmark Bandpass

Unità Funzionale	Vincolo	LB	FD
indirect_ref	inf	13	6
MINUS	2	2	2
MUL	1	1	1
PLUS	2	1	2
Tempo computazione		0.22	75.77

Tabella 6.10. Unità allocate per il benchmark Bandpass

6.6.2.2 Benchmark Diffeq

Unità Funzionale	Vincolo	LB	FD
ASSIGN	inf	1	1
CMP	2	1	1
indirect_ref	inf	7	3
MINUS	2	1	1
MUL	1	1	1
PLUS	2	1	1
READ_COND	inf	1	1
Tempo computazione		0.10	0.99

Tabella 6.11. Unità allocate per il benchmark Diffeq

6.6.2.3 Benchmark Ewf_v2

Unità Funzionale	Vincolo	LB	FD
indirect_ref	inf	11	5
MUL	1	1	1
PLUS	2	2	2
Tempo computazione		0.25	10.04

Tabella 6.12. Unità allocate per il benchmark Ewf_v2

Capitolo 7

Conclusioni e possibili sviluppi futuri

In questo lavoro di tesi è stato proposto un possibile algoritmo di scheduling basato sul Force Directed che consente di cercare di minimizzare il numero di unità funzionali allocate in un sistema estraendo il massimo parallelismo possibile dalla specifica del sistema descritto attraverso un SDG. A differenza degli algoritmi tradizionali, il metodo proposto permette di considerare contemporaneamente vincoli temporali e vincoli relativi alle risorse. Questa completezza è pagata in termini di tempi di computazione che rispetto ad altri algoritmi più semplici sono notevolmente maggiori. Tuttavia considerando i guadagni non trascurabili in termini di area e conseguentemente in termini di potenza dissipata si può essere disposti a pagare questo tempo in fase di progettazione del sistema.

I risultati dei test proposti hanno dimostrato la bontà delle modifiche effettuate e l'utilità nell'aver creato un algoritmo che gestisca contemporaneamente vincoli temporali e sulle risorse cercando di minimizzare una metrica relativa al costo dell'implementazione.

La versione dell'algoritmo presentata si è quindi dimostrata sufficientemente completa e potrebbe anche essere considerata come definitiva. Esistono tuttavia ancora dei possibili margini di miglioramento e delle possibili estensioni. Infatti ci sono ancora degli aspetti secondari rimasti insoluti all'interno

del lavoro presentato che potrebbero costituire l'oggetto di futuri lavori:

- individuare un criterio più accurato per assegnare la priorità alle operazioni, in particolare determinando il criterio per l'ordinamento totale dei blocchi basici che fornisca i migliori risultati in termini di unità funzionali allocate e tempo di computazione, ed eventualmente introdurre la possibilità di assegnare priorità diverse alle operazioni appartenenti ad uno stesso blocco basico (cioè schedulare un blocco basico a pezzi invece che interamente), in modo tale da ridurre il tempo di computazione anche in problemi privi di costrutti condizionali;

- estendere l'algoritmo modificato per minimizzare altri aspetti del costo relativo all'implementazione quali numero di registri o costo delle interconnessioni;

- individuare come inserire all'interno dell'algoritmo informazioni relative al diverso costo delle unità funzionali in presenza di vincoli; in questo caso infatti l'algoritmo che è stato presentato cerca di minimizzare il numero complessivo di unità funzionali appartenenti a tipi non vincolati indistintamente; sicuramente però si vorrebbe che la minimizzazione del numero di risorse non vincolate di tipo più costoso fosse favorita rispetto a quella delle unità più economiche; l'algoritmo originale veniva indirizzato in questo senso attraverso pesi, calcolati sulla base del costo delle unità funzionali, attribuiti alle diverse somme di probabilità; è evidente che lo stesso metodo deve essere applicato anche nella versione proposta, ma sussiste il problema di come combinare i pesi derivanti dal costo delle unità funzionali con quelli relativi ai vincoli;

- valutare se vi sia un effettivo miglioramento nella qualità dei risultati introducendo un calcolo meno approssimativo delle *predecessors' and successors' forces* che tenga conto delle dipendenze indirette fra le riduzioni delle mobilità di operazioni contemporanee che sono predecessori o successori dell'operazione di cui si sta esaminando l'assegnamento;

- soprattutto da un punto di vista implementativo, considerare la possibilità di ridurre il tempo di computazione dell'algoritmo, nel caso di presenza

di vincoli, a scapito della memoria da esso utilizzata: tenendo memorizzati i dati relativi ad ogni nodo esaminato nell'esplorazione dell'albero di ricerca è possibile ridurre il numero di percentuali di occupazione o di forze da ricalcolare in caso di backtracking.

Riferimenti bibliografici

[App98] Andrew W. Appel, *Modern compiler implementation in java*, Cambridge University Press, 1998.

[Bel71] Alan Newell C. Gordon Bell, *Computer structures: Readings and examples*, McGraw-Hill, 1971.

[BeMR75] JamesR. Bitner and edward M. Reingold, *Backtrack programming techniques*, Communications of the ACM **18 (11)** (1975), 651–656.

[boo06] *boost c++ libraries*, www.boost.org, 2006.

[BP91] Raul Camposano Reinaldo A. Bergamaschi and Micheal Payer, *Area and performance optimizations in path–based scheduling*, IEEE (1991), 304–310.

[Bra05] Luca Brambilla, *Una metodologia per la sintesi ad alto livello dell'unitá di controllo di un sistema dedicato*, Master's thesis, Politecnico di Milano, 2005.

[Cam90] Reinaldo A. Bergamaschi Raul Camposano, *Synthesys using path based scheduling*, ACM/IEEE Design Automation Conference, vol. 25, 1990, pp. 450–455.

[CD86] C.J.Tseng and D.P.Siewiorek, *Automatic synthesis of data path on digital systems*, IEEE Transaction on Computer - Aided Design of Integrated Circuits and Systems, vol. 5, July 1986, pp. 379–395.

[CM87] C.H.Genotys and M.I.Elmasry, *A vlsi methodology with testability constraints*, Proc. 1987 Canadian Conference VLSI, October 1987.

[DLSM81] Scott Davidson, David Landskov, Bruce Shriver, and Patrick W. Mallett, *Some experiments in local microcode compaction for horizontal machines*, IEEE Transactions on Computer C-32, vol. 7, July 1981, pp. 460–477.

[ea92] D.Gajski et al., *High level synthesis - introduction to chip and system desing*, Kluwer Academic Publisher, 1992.

[Fou06] Free Software Foundation, *Gnu gcc*, http://gcc.gnu.org, 2006.

[FS] Franco Fummi and Mariagiovanna Sami, *Il progetto funzionale: l'approccio controlore-datapath*.

[GB65] Solomon W. Golomb and Leonard D. Baumert, *Backtrack programming*, Journal of the Association for Computing Machinery **12 (4)** (1965), 516–524.

[GR94] Daniel D. Gajski and Loganath Ramachandran, *Introduction to high-level synthesis*, IEEE Design and Test of Computers **11** (1994), no. 4, 44–54.

[HT83] C.Y. Hitchcock and D.E. Thomas, *A method of automatic data path synthesis*, Proceedings of the Design Automation Conference (DAC), June 1983, pp. 484–489.

[ICCM91] I-C.Park and C-M.Kyung, *Fast and near optimal scheduling in automatic data path synthesis*, Proc.of the 28th DAC, 1991, pp. 680–685.

[JY89] Y.Hsu J.Lee and Y.Lin, *A new integer linear programming formulation for scheduling problem in data–path synthesis*, Proc.of the Int.conf.on Computer–Aided Design, 1989, pp. 20–23.

[KP90] K. Kucukcakar and A.C. Parker, *Data path tradeoff using mabal*, Proceedings of the Design Automation Conference (DAC), 1990, pp. 511–516.

[KS70] K.H.Kernighan and S.Lin, *An efficient heuristic procedure for partitioning graphs*, Bell system technical journal, vol. 49, pp. 291–307, February 1970.

[mlDdEeIPdM06] μ-lab Dipartimento di Elettronica e Informazione Politecnico di Milano, *Panda project*, http://savane.elet.polimi.it/panda/index.shtml, 2006.

[Moo65] Gordon E. Moore, *Cramming more components onto integrated circuits*, IEEE Design and Test of Computers **38** (1965), 114–117.

[PK87] P. G. Paulin and J. P. Knight, *Force-directed scheduing in automatic data path synthesis*, Proceedings of the 24th ACM/IEEE conference on Design automation, ACM Press, 1987, pp. 195–202.

[PK89] _____, *Force-directed scheduling for the behavioral synthesis of asic's*, IEEE Transaction on CAD, vol. CAD-8, july 1989, pp. 671–679.

[PKG86] P. G. Paulin, J. P. Knight, and E. F. Girczyc, *Hal: A multi-paradigm approach to automatic data path synthesis*, Proceedings of the 24th ACM/IEEE conference on Design automation, July 1986.

[RH91] A.Sharma R.Jain, A.Mujumdar and H.Wang, *Empirical evaluation of some high–level synthesis scheduling heuristics*, Proc.of 28th DAC, 1991, pp. 210–215.

[SA89] S.Devadas and A.R.Newton, *Algorithm for allocation in data-path synthesis*, IEEE Trans.on CAD of Integ.Cir.and System, vol. 8, July 1989, pp. 768–781.

[Sys] *Open systemc initiative (osci)*, http://www.systemc.org.

[Wal60] R. J. Walker, *An enumerative techinque for class of combinatorial problems*, Combinatorial Analysis (Proceedings of Symposium in Applied Mathematicc) (Providence, Rhode Island), vol. X, American Mathematical Society, 1960, pp. 91–94.

Ringraziamenti

Il primo ringraziamento e il più doveroso è per i miei genitori che hanno sopportato la mia parte studentesca (c'è una parte studentesca in tutti) per ben diciannove anni ed è un grosso merito perchè durante i periodi di studio-compiti-esami credo di essere stato abbastanza insopportabile. Con il loro supporto mi hanno dimostrato che molteplici e opposti possono essere i modi per spingere una persona verso il suo obiettivo. Voglio poi dedicare un ricordo ai miei nonni perchè sicuramente è anche merito loro se sono diventato quello che sono.

Ringrazio poi il Prof. Fabrizio Ferrandi: non solo per il supporto e l'aiuto datomi durante il lavoro di tesi, ma soprattutto perchè è grazie agli insegnamenti dei suoi corsi se all'interno del mare magnum dell'informatica mi sono appassionato alla progettazione di sistemi dedicati ed ho quindi continuato il percorso che mi ha portato a questa tesi di laurea. Ironicamente devo anche ringraziare chi involontariamente mi ha spinto a provare a percorrere questa strada imponendo certe scelte nei piani di studio.

Estendo quindi il ringraziamento a tutti quei professori, ricercatori e dottorandi che ho incontrato durante questi sei anni di studio al Poli: a ciascuno di loro devo un pezzetto più o meno grande di questa laurea e di quella del primo livello. A questo punto dovrei ringraziare anche i miei colleghi con cui ho fatto qualche progetto ma lo farò in seguito perchè sono prima amici che colleghi.

Ringrazio quindi tutti quelli con cui ho percorso un tratto più o meno breve di questi anni, a partire dagli amici con cui ho condiviso i primi semestri e di cui ormai a distanza di cinque anni ricordo più il soprannome che il nome: il mio "quasi parente" Edo (chissà se sei poi andato in Scandinavia), Alain (spero si scriva così), Matteo, Francesco, Silvia, il capozarro, Pocchio e altri che probabilmente in questo momento dimentico. Un particolare ringraziamento a Lucky con cui ho condiviso oltre alla vita dentro e fuori l'università la tragica esperienza del pendolarismo ferroviario; il pensare che sono ormai undici anni che ci conosciamo mi fa rendere conto di quanto siamo ormai "vecchi" (tu resti comunque più vecchio di ben una settimana). Ecco, a proposito di pendolarismo, NON ringrazio le Ferrovie dello Stato per tutte le ore perse in questi anni, i tentativi andati a vuoto di farmi mancare un esame e per i numerosi tentativi di

farmi ammalare tramite caldo, freddo e mancanza di ossigeno. E' stata una dura prova di sopravvivenza, ma sono riuscito a uscirne abbastanza indenne (ma rimane ancora il giorno della laurea da affrontare).

Ringrazio quindi tutti gli amici-colleghi con cui ho condiviso il percorso di questa Laurea Specialistica all'interno del percorso degli esami "HW-SW", fra cui soprattutto Dario anche per l'esperienza del lavoro svolto insieme, e Miele che è stato molto di più che un semplice compagno di progetto e gli amici del Laboratorio di Ing. Software.

Ho voluto lasciare per ultimi i ragazzi con cui maggiormente ho condiviso questa grande esperienza che è la vita universitaria (che è ben diverso dal condividere l'università!) e cioè i ragazzi della mailingleet (facciamo un po' di pubblicità che non fa mai male www.leet.it; pubblicità a cosa? andate a vederlo). Grazie a loro per avermi fatto vivere e contribuire a formare la filosofia del *lost* e del *leet*. "Losto" la sua descrizione perchè ci vorrebbe una tesi solo per questo. Fra i tanti del gruppo in particolare voglio ricordare il Mazzu e la sua pazzia matematica (temo sia definitivamente irrecuperabile), il Lazza e il suo TAF, il Bongo (sono io o sei tu che si deve preoccupare del fatto che appena ho pensato al Lazza ho pensato anche a te?), quella fonte infinita di lost che è Giani, il Plinio che ci ha sempre mostrato come un ingegnere informatico debba vivere, Carlo che invece ci ha sempre mostrato come un ing. debba realizzare un progetto, gli interisti (Maio, Claudio e il Berti) con cui ho discusso di un calcio che si è poi rivelato falso, Geo con cui condivido la fede calcistica (Forza Milan!!), il professor "Lale" e poi Domenico, CareLuca, Teo Parabiago, il Ninja, Michele, Aladino (lo so che sembra strano, ma non è un soprannome) e tutti gli altri iscritti alla mailing.

Dimentico qualcuno? Probabilmente sì perchè in questi giorni ormai sono abbastanza fuso. Se ciò dovesse essere accaduto me ne scuso con gli interessati. Grazie a tutti!