Estratto da

R. Ferro e A. Zanardo (a cura di), *Atti degli incontri di logica matematica* Volume 3, Siena 8-11 gennaio 1985, Padova 24-27 ottobre 1985, Siena 2-5 aprile 1986.

Disponibile in rete su http://www.ailalogica.it

PROGRAMMAZIONE E LOGICA: ESPERIENZE E ASPETTATIVE

INTERVENTO DI
P.G. BOSCO, G. GIANDONATO, S. GIORCELLI, E. GIOVANNETTI, S. SOFI
CSELT, Torino

ASPETTI DI UNA RICERCA CSELT SU MACCHINE E LINGUAGGI DI NUOVA GENERAZIONE

Sommario

Nell'articolo sono descritti gli indirizzi di un programma di ricerca su macchine e linguaggi di nuova generazione, fondati sulla programmazione logica. Tali indirizzi riguardano, in particolare: lo sviluppo di un'architettura parallela MIMD basata su un insieme di nodi realizzati ciascuno attorno ad un Transputer a 32 bit, e connessi da una rete multistadio a pacchetto di tipo Delta; lo studio di modelli computativi paralleli per il Prolog appropriati per applicazioni di IA in tempo reale; la definizione di un linguaggio integrato logico e funzionale e di un corrispondente modello esecutivo; l'utilizzo della programmazione logica nel campo dei linguaggi di specifica per la descrizione e il progetto di grossi sistemi di telecomunicazioni.

1. Introduzione

Questo lavoro descrive gli indirizzi di un programma di ricerca attivo da circa due anni in CSELT su macchine e linguaggi cosiddetti di nuova generazione ed attinente dunque per larga parte alle tematiche della programmazione logica. Questa non e' pero' assunta come paradigma iniziale o di riferimento della ricerca che, piuttosto, e' volutamente pilotata dalle applicazioni e tesa ad accertare se e come su problemi specifici ma reali e di grossa taglia (ad es. lo studio sintattico-semantico dei sistemi di comprensione del linguaggio naturale parlato) le promesse accompagnatesi alla rapida crescita d'interesse e di attivita' sull'elaborazione di nuova generazione siano, almeno in parte, mantenibili.

Lo spazio in cui si muove questa attivita' e' dunque quello, tipico per CSELT, a meta' strada fra la ricerca di base, a contenuti prevalentemente teorici come nelle Universita', la cui collaborazione risulta in questo caso e proprio per via di tali contenuti essenziale sugli aspetti piu' avanzati del programma (vedi l'esempio, piu' avanti, di cooperazione nel progetto su linguaggi logico-funzionali), e la ricerca-sviluppo dei laboratori industriali, direttamente mirata ai possibili prodotti. In particolare per i temi affrontati questo spazio e' tuttora ritenuto molto vasto e tale da giustificare approcci pragmatici (come nel caso delle architetture ad elevato parallelismo, vedi par. 2), o una diversificazione degli sforzi sia nel

tempo sia nella dimensione tecnica degli obbiettivi (come per i linguaggi di programmazione non-convenzionali, par. 3), o, infine, una fase di studio preliminare (e' il caso dell'applicazione all'ambiente del software di telecomunicazioni di linguaggi formali di specifica, par.4).

2. Architetture parallele

Come e' noto, una delle piu' rilevanti promesse della elaborazione di nuova generazione consiste nella intravista possibilita' di sfruttare l'elevato grado di parallelismo di architetture fisiche quali quelle rese fattibili dallo sviluppo delle tecnologie VLSI mediante stili di programmazione dichiarativi, a semantica non intrinsecamente sequenziale perche' basati su "regole" (di riscrittura, di implicazione logica, di produzione, e di comportamento, come nei linguaggi funzionali, in quelli logici, nei sistemi di inferenza forward, e nei linguaggi di programmazione concorrente tipo CP, rispettivamente). Trattandosi di macchine orientate al linguaggio l'approccio piu' corretto in linea di principio sarebbe quello ("language first") di far discendere il disegno dell'architettura dal modello computazionale e questo dalla semantica operazionale del linguaggio non-convenzionale adottato. In assenza, come nel nostro caso, di un unico linguaggio di riferimento ed anzi essendo obbiettivo della ricerca l'individuazione per un tale linguaggio delle primitive di controllo del parallelismo piu' adatte alla classe di problemi affrontata, l'approccio alla definizione dell'architettura fisica (qui distinta da quella virtuale che su di essa implementa l'esecutore parallelo) e' dal basso ossia e' derivata da vincoli di tipo tecnologico, principalmente legati all'ottica VLSI, e da caratteristiche ritenute comuni alla varieta' dei modelli computativi indagati.

La macchina in progetto, di cui e' previsto un prototipo con grado di parallelismo sufficiente per una realistica sperimentazione (64-128 processori), si basa in particolare sulla eliminazione di memoria comune, sostituita dalla possibilita' di uno schema di indirizzamento globale ad una memoria omogeneamente distribuita sugli elementi computativi (il motivo e' ovviamente legato all'esigenza di omogeneita', essenziale nella prospettiva tecnologica di lungo termine in cui ogni nodo processore + modulo di memoria + elemento di comunicazione sara' realizzabile con pochissimi circuiti); sul privilegiamento di modelli computazionali con granularita' media (ossia tali da allocare in sequenza sul singolo processore una quantita' significativa di istruzioni da eseguire per ogni unita' di lavoro schedulata; qui il motivo e' rintracciabile nella necessita' di evitare eccessivi overheads nello sfruttamento del parallelismo e nell'obbiettivo di assicurare su ogni nodo del sistema parallelo l'efficienza ottenuta sul processore sequenziale mediante le tecniche convenzionali, ad es. compilative, disponibili per esecutori control-flow); su uno schema di comunicazione interprocessor, basato su commutazione veloce di pacchetto, consistente con i requisiti di scalabilita' (scegliendo opportune topologie di rete), relativa insensibilita' ai ritardi (mediante elementi computativi caratterizzati da rapidissimo context-switching) e ottimizzazione della localita'.

Il prototipo target e' costituito da un insieme omogeneo di nodi costruiti ciascuno attorno ad un Transputer a 32 bit e connessi da una rete multistadio a pacchetto di tipo Delta, realizzata mediante un circuito integrato custom, attualmente in fase avanzata di sviluppo presso CSELT. Una rete statica, realizzata direttamente con i link disponibili sul Transputer, permette l'ottimizzazione, ad es.

ai fini del bilanciamento di carico, della comunicazione strettamente locale, di tipo nearest-neighbours, ed il raccordo con l'elaboratore host, un Microvax II, per caricamento, collezione dei risultati e monitoraggio del sistema. Nella nostra ricerca il Transputer e' considerato come una "approssimazione" dell'elemento computativo ideale, e tra gli obbiettivi collaterali del programma si colloca anche quello di derivare le caratteristiche ulteriori che un processore di questo tipo (ossia RISC, con comunicazione e multitasking built-in, a veloce process-switching) dovrebbe possedere per risultare ottimale nelle applicazioni di elaborazione simbolica.

3. Linguaggi di programmazione

Un tipico problema che richiede un elevato grado di parallelismo nell'elaborazione simbolica e' il processo di comprensione del linguaggio naturale parlato ai livelli di trattamento sintattico, semantico e pragmatico, dove l'incertezza dei dati in ingresso (nel sistema attualmente in sviluppo presso CSELT un database, proveniente dal precedente stadio di riconoscimento del segnale, di ipotesi lessicali concernenti la probabile presenza di una parola in un intervallo temporale), accompagnata al gran numero di regole e fatti coinvolti nell'analisi ed all'esigenza di tempo reale imposta dalle possibili applicazioni, genera requisiti di throughput incompatibili con le tecniche elaborative convenzionali. In termini di programmazione logica il problema puo' essere raffrontato con quello, storicamente legato al Prolog fin dalla sua origine (vedi DCG e simili), della analisi di linguaggio naturale, con la differenza essenziale pero' che nel caso del parlato sono richieste strategie di controllo nella generazione e nell'esplorazione dello spazio di ricerca ben piu' complesse (ad es. parsificazione non left-to-rigth, mista top-down e bottom-up, con memorizzazione dei risultati intermedi, guidata da euristiche programmabili del tipo best-first, etc.).

Il punto di partenza, e di riferimento per accertare i vantaggi di un approccio basato sulla programmazione logica, e' l'implementazione di una versione parallela degli algoritmi di comprensione mediante un ambiente di processi LISP concorrenti e comunicanti a messaggi asincroni. Gia' in tale ambiente, che e' in corso di sviluppo sulla macchina sopra descritta, sono stati notati i vantaggi espressivi che deriverebbero dall'aver integrata nella parte funzionale del LISP un'efficiente componente logica interfacciata al database di fatti e regole che costituiscono la base di conoscenza del singolo processo.

L'obbiettivo e' la riformulazione in termini di programmazione logica degli stessi algoritmi al fine di una valutazione comparata dei benefici e degli svantaggi dei due approcci. Appare evidente, dagli accenni di cui sopra, come estensioni del Prolog, contemporaneamente adatte alla specificita' del problema e ad un'efficiente realizzazione parallela, siano necessarie e come d'altra parte esse non siano banali, sia dal punto di vista concettuale sia implementativo (si pensi ad una "assert" in architetture distribuite).

Un primo passo della ricerca e' consistito nell'analisi e nell'organizzazione il piu' sistematica possibile dei modelli computativi paralleli per il Prolog. Quattro modelli sono stati scelti fra i piu' significativi proposti recentemente in

letteratura (quello di Conery, di Haridi e Ciepielewski, di Lindstrom, ed il PIE) e lo sforzo e' stato quello di confrontarli e correlarli in modo da ottenere un quadro generale dei problemi da affrontare e delle tecniche che si possono utilizzare per l'implementazione di un linguaggio logico. In particolare sono stati esaminati a fondo i seguenti aspetti:

- forma e grado di parallelismo. Come e' noto, Conery ha distinto quattro forme di parallelismo (OR, AND, stream, search), ma in realta' questa classificazione risulta grossolana per i nostri scopi; ad es. all'interno della forma OR diversi gradi di parallelismo sono possibili ("pipelining", "backtracking free", ecc.), corrispondentemente a differenti primitive di controllo richieste come estensione al linguaggio.

- rappresentazione dei dati. Dal nostro studio e' emerso che il classico problema copying/sharing ha tre diversi aspetti, che riguardano rispettivamente la rappresentazione di "goal", "skeleton" ed "environment". In particolare in un ambiente parallelo e' di importanza vitale trovare una soluzione accettabile per l'ultimo aspetto (environment copying/sharing). Una soluzione ottimale e' strettamente dipendente non solo dall'architettura fisica (ad es. memoria centralizzata oppure distribuita), ma anche dal tipo di applicazione. Comunque sembrano attraenti, in quanto buon compromesso, strategie miste basate sul "copying incrementale" ed il "lazy fetch".

- conflitto "OR". Un problema tipico nell'implementazione del Prolog e' dato dal cosiddetto "conflitto OR", strettamente connesso al nondeterminismo "don't know" del linguaggio. Un interprete sequenziale risolve tale conflitto grazie ai concetti di: unico "binding environment", attraversamento in profondita' dell'albero di ricerca, procedura di "backtracking", "trail list". Per un'esecuzione OR-parallela invece sono stati introdotti altri metodi, riconducibili a due tipi di meccanismi base: replica ad ogni nodo nondeterministico delle variabili ancora non istanziate (si vedano ad es. il "binding array" di D.S. Warren e la recente implementazione del CP di Tacheuchi); "unificazione in due fasi" (Wise, Lindstrom), secondo la quale parte delle assegnazioni dovute all'unificazione (e precisamente quelle relative alle variabili del goal corrente) non sono eseguite all'atto della chiamata della clausola, ma rinviate alla seconda fase, cioe' quando la clausola e' conclusa ed il controllo ritorna al goal corrente. La scelta fra questi due metodi dipende fortemente dal tipo e dalla forma di parallelismo che si intende supportare ed inoltre e' in stretta correlazione con la questione copying/sharing illustrata precedentemente.

Un tema di ricerca connesso con i precedenti, ma piu' a lungo termine, e motivato piu' in generale dalla necessita', in molte applicazioni, di tipi diversi di elaborazione, cioe' di una parte procedurale/algoritmica, tipicamente deterministica, e di una parte dichiarativa/inferenziale, tipicamente nondeter- ministica, le quali sono molto naturalmente esprimibili rispettivamente in un linguaggio funzionale e in un linguaggio logico, e' quello che viene affrontato dallo CSELT, unitamente all' Universita' di Pisa, all'interno di un progetto ESPRIT nel sottoprogetto "Integration of logic and functional languages", che ha come obiettivi dapprima la definizione di un linguaggio unitario in cui i due stili di programmazione rispettivamente logico e funzionale siano integrati in modo "profondo", cioe' con una semantica chiara e ben definita, in contrapposizione al semplice interfacciamento di due linguaggi distinti; poi la definizione di un modello computazionale che tratti anch'esso in modo unitario i due aspetti del linguaggio, e che contemporaneamente sfrutti l'elevato grado di parallelismo che, come si e' detto, e'

oggi possibile realizzare su un'architettura fisica.

L'attivita' in CSELT in questo primo anno del progetto e' stata rivolta soprattutto all'analisi del significato dell'integrazione logico-funzionale dal punto di vista della logica del prim'ordine con uguaglianza, in particolare della logica delle clausole di Horn con uguaglianza. Tale indagine ha messo in evidenza, da un lato, la continuita' concettuale dai sistemi di inferenza generali per la logica con uguaglianza (ad esempio risoluzione piu' paramodulazione) fino ai sistemi inferenziali specializzati adottabili come interpreti di linguaggi di programmazione. D'altro lato ha posto in risalto proprio il fatto che sistemi generali, utilizzabili come dimostratori di teoremi, sono invece poco adatti a servire di base per interpreti di veri linguaggi di programmazione, in cui si vuole che la computazione non sia semplicemente ricerca in uno spazio di possibili soluzioni, ma resti invece ancora in qualche modo sotto il controllo del programmatore. Piu' in concreto, si tratta dell'esigenza che l'algoritmo di inferenza sia "lineare" nello stesso senso in cui lo e' la risoluzione SLD che sta alla base del Prolog; cio' equivale a richiedere che si prendano in considerazione soltanto classi di "programmi" per le quali l'algoritmo di soluzione di goals fondato su risoluzione lineare e narrowing - o, equivalentemente, solo sulla risoluzione, previo "appiattimento" del programma - sia completo. Si esclude cosi' il ricorso ad algoritmi di completamento alla Knuth-Bendix e simili, tranne eventualmente in una fase di "compilazione", di cui pero' non e' in generale garantita la terminazione, data la natura semidecidibile del problema.

Uno dei principali nodi da affrontare a livello di definizione del linguaggio e' costituito dall'integrazione nella cornice di cui sopra delle funzioni di ordine superiore, che costituiscono una caratteristica essenziale dei linguaggi funzionali moderni. A questo riguardo un'attraente direzione potrebbe risultare quella indicata dai lavori di Martin-Löf e di molti altri che si fonda sulla matematica costruttiva e sulla logica intuizionistica.

4. Linguaggi di specifica

Da anni lo CSELT e' attivo nel campo dei linguaggi di specifica per la descrizione e il progetto di complessi sistemi di telecomunicazioni. Una intensa attivita' si e' sviluppata in ambito C.C.I.T.T. per la definizione di un linguaggio di specifica (SDL) capace di trattare aspetti sequenziali, concorrenti e di strutturazione; parallelamente in CSELT si porta avanti una ricerca di maggiore generalita' su tali temi e con maggiore enfasi sui tools di supporto a tali linguaggi.

Dal punto di vista della definizione di linguaggi di specifica l'aspetto logico ha fatto il suo ingresso a livello C.C.I.T.T. nella formulazione della nuova proposta Z104 (la componente "abstract data type" dell'SDL) che con un forte contributo di CSELT-SIP e' stata indirizzata verso una forma di clausole di Horn con uguaglianza (tipo EQLOG). Si prevede che tale aspetto sara' ulteriormente enfatizzato nel periodo successivo (Questione II).

Riguardo all'utilizzo specifico di programmazione logica, si puo' dire che in generale si utilizza il Prolog come linguaggio di manipolazione simbolica, in

alternativa al LISP, in quelle attivita' in cui nondeterminismo e unificazione sono elementi caratteristici del problema. In particolare:

a) Definizione di semantiche operazionali con regole d'inferenza ('a la Plotkin') dei linguaggi di specifica esistenti. Con questo metodo si riescono ad ottenere in breve tempo dei simulatori piu' o meno simbolici (graph builders) per i linguaggi d'interesse. Il meccanismo di backtracking semplifica notevolmente l'analisi del nondeterminismo don't care presente nelle parti concorrenti.

b) Implementazione di procedure di decisione per logiche particolari (ad esempio temporali). Il Prolog viene utilizzato per l'implementazione delle componenti "speciali" delle logiche in questione, ad esempio attraverso l'esplicitazione della loro definizione semantica tramite predicati del tipo di "holds". La componente logica del prim'ordine viene in genere demandata al Prolog stesso.

c) Strumenti di supporto alla specifica con Abstract Data Types. Il Prolog viene utilizzato con l'ottica "logica" nella conversione di regole di riscrittura in clausole che assiomatizzano un certo predicato di uguaglianza come linguaggio di programmazione per l'implementazione di procedure di riscrittura, narrowing, completamento. In taluni casi ci si scontra con l'assenza dell'occur check (ad esempio nel calcolo delle coppie critiche nell'algoritmo di Knuth-Bendix), che deve pertanto essere programmato in Prolog con ovvie inefficienze Vengono anche implementati algoritmi speciali di unificazione (C, AC) che hanno come base l'unificazione standard. Il Prolog fornisce gia' un ambiente in cui non ci si deve preoccupare della rappresentazione dei bindings, delle varibili, etc.

d) Fast Prototyping, sperimentazione di nuovi linguaggi e integrazione di aspetti funzionali e logici. Un interessante esperimento ha condotto alla costruzione di un mini-ambiente funzionale (tipo ML). L'insieme di type-checker e traduttore che e' stato sviluppato in tre giorni e ammonta a circa 5 pagine Prolog e' in grado di trattare "efficientemente" gli oggetti funzionali sfruttando l'idea originale di Warren.

5. Note conclusive

Naturalmente l'esposizione qui riportata non e' esaustiva delle applicazioni o degli interessi connessi alla programmazione logica presenti in CSELT, dove esperienze significative, ad es. nel campo dei sistemi esperti, sono state maturate, ed altre sono in corso.

Il lavoro descritto e' parzialmente finanziato dai progetti ESPRIT N.26 (Advanced Algorithms and Architectures for Signal Recognition and Understanding) cui collaborano gruppi del Dipartimento di Informatica dell'Universita' di Torino, e N.415 (Parallel Architectures and Languages for Advanced Information Processing), al quale lo CSELT partecipa in cooperazione con il Dipartimento di Informatica dell'Universita' di Pisa.