Architettura IA−64

Davide Chiarella, Francesco Figari, Daniele Venzano

L'IA−64 è la nuova architettura a 64 bit di Intel e Hp. È stata progettata con un'estesa capacità architetturale: spazio di indirizzamento a 64 bit, elevato numero di registri ed elevato numero di bit per istruzione. Si è inoltre voluto mantenere la retrocompatibilità con l'IA−32. Per aumentare l'Instruction Level Parallelism il linguaggio macchina permette al compilatore di creare gruppi di istruzioni da eseguire in parallelo. L'IA−64 realizza inoltre la filosofia EPIC (Explicit Parallel Instruction Computing) poiché fornisce al compilatore meccanismi software per organizzare la computazione basandosi sulla sua conoscenza globale ed un set di istruzioni che permette di comunicare all'hardware le informazioni chiave che ha raccolto durante l'analisi del programma.

Ogni istruzione dell'IA−64 è ampia 41 bit (14 bit per l'operazione, 3 operandi da 7 bit e 6 bit per il registro predicato).

Op	Register 1	Register 2	Register 3	Predicate
14 bits	7 bits	7 bits	7 bits	6 bits

Le istruzioni vengono codificate in *bundle* di 128 bit: 3 istruzioni (per un totale di 123 bit), più un *template* di 5 bit che specifica il mapping fra gli instruction slot e le Execution Unit.

Instruction 2	Instruction 1	Instruction 0	Template
41 bits	41 bits	41 bits	5 bits

L'IA−64 supporta tre modalità di funzionamento: con sistema operativo a 32 bit, con sistema operativo a 64 bit o a 64 bit che esegue applicazioni per IA−32.

Registri e loro funzioni

- 128 registri general−purpose (GR) a 64 bit (+1 bit NaT)
  - ♦ 32 sono utilizzabili in modalità IA−32
  - ♦ 96 sono rinominabili via software per realizzare stack e software pipelining
- 128 registri floating point (FR) a 82 bit, con le stesse caratteristiche del punto precedente
- 64 registri da 1 bit per i predicati (PR)
- 8 registri di branch da 64 bit (BR)
- 1 registro CFM (Current Frame Marker) che mantiene lo stato dello stack frame corrente e della rotazione dei registri durante i loop (38 bit)
- 128 Application Registers per il supporto alla gestione della comunicazione tra sistema operativo e applicazioni utente

Supporto al Sistema Operativo

Lo spazio di indirizzamento in un’architettura a 64 bit è molto vasto e non è possibile utilizzarlo come su un’architettura a 32 bit perché le diverse tavole di traduzione tra indirizzi logici e fisici sarebbero troppo grandi. L'IA−64 offre la possibilità di usare pagine di dimensione variabile, fino a 256 MB (su IA−32 le pagine hanno dimensione 4 KB). Inoltre sono disponibili alcuni registri che permettono di suddividere lo spazio di indirizzamento in 8 aree che il sistema operativo può utilizzare per scopi differenti (dati statici, memoria condivisa, librerie dinamiche, ...).

Stack sui Registri

I 96 registri GR e FR e i 48 registri PR rinominabili sono utilizzati per mantenere all'interno della CPU più di uno stack frame alla volta: al momento della chiamata di una procedura i registri vengono rinominati in modo da lasciare al chiamato un set pulito di registri. Nel momento in cui non c'è più spazio per un nuovo frame, una parte dei registri viene ricopiata in memoria in maniera trasparente all'utente. Questo meccanismo consente, in media, di ridurre del 30% gli accessi in memoria.

Software Pipelining

Il problema è sfruttare al meglio le risorse di esecuzione parallela durante i loop. Per determinati tipi di loop è possibile inserire le singole iterazioni in una pipeline virtuale, facendo iniziare l'iterazione successiva prima che quella precedente sia terminata. L'IA−64 offre il pieno supporto a questo meccanismo fornendo istruzioni apposite e registri dedicati in modo da ridurre la dimensione del codice che il compilatore deve generare per gestire l'esecuzione parallela senza che due iterazioni interferiscano tra loro.

Istruzioni Predicated

Con *predication* si intende l'esecuzione condizionale di una singola istruzione, dove il valore della guardia è dato da un registro per predicato. Il set di predicate register introdotto dalla nuova architettura è costituito da 64 registri da 1 bit, in cui vengono memorizzati i risultati delle istruzioni *test* e *compare*, nelle loro varianti. Test e compare lavorano su coppie di registri: in uno viene scritto il risultato dell'operazione, nell'altro il calore negato. Questo permette di utilizzare l'esecuzione predicata per codificare con un numero minimo di istruzioni strutture *if−then−else*, multiway branch e confronti paralleli.

Branch Prediction

La struttura a pipeline rende necessario un meccanismo di predizione dei salti: gli stadi iniziali della pipeline vengono riempiti con le istruzioni del ramo del codice che ha maggior probabilità di essere preso. Oltre alle istruzioni predicate, che permettono di eliminare alcuni salti al costo di poche no−op, l'IA−64 mette a disposizione degli strumenti per inserire nel codice *indizi* sui salti condizionali che l'algoritmo di branch prediction implementato nel processore puo' utilizzare nelle sue scelte. Questi indizi possono essere forniti attraverso modificatori delle normali istruzioni di salto condizionale oppure con separate istruzioni *brp*, che permettono di specificare anche il tipo di predizione da utilizzare.

Indizi disponibili per indirizzare la branch prediction:

- Prediction Strategy
  - Taken / Not−Taken: il salto è di solito probabile / improbabile
  - Static / Dynamic: i salti statici vengono sempre predetti con il valore di Taken / Not−Taken; per quelli dinamici si applica l'algoritmo di branch prediction, utilizzando Taken / Not−Taken quando non ci sono informazioni storiche
- Sequential Prefetch
  - Few / Many: indica la quantità di istruzioni da precaricare lungo il ramo predetto (il valore numerico esatto dipende dall'implementazione dell'architettura)
- Predictor Deallocation
  - Clr: permette di azzerare le informazioni sul salto mantenute dall'algoritmo di branch prediction, che altrimenti utilizza una politica MRU

Compiler−directed Speculation

Dal punto di vista del compilatore, al momento dell'ottimizzazione un programma è una sequenza di basic

block. All'interno del grafo che lo rappresenta esiste un percorso che ha maggior probabilità di essere eseguito; questo cammino può essere considerato come un solo blocco e le sue istruzioni riordinate ed eseguite in parallelo. Spostare il codice lungo il grafico dei basic block può infrangere però due tipi di dipendenze: quelle relative al flusso di controllo (e.g. eseguire un'operazione prima di un salto condizionale) e quelle relative a letture e scritture in memoria (e.g. leggere un dato prima di una sua possibile riscrittura). In questo caso si parla di *speculation* (esecuzione speculativa): si esegue del codice che 'probabilmente' sarà utile. L'IA−64 supplisce strutture per entrambi i tipi di speculazione.

Control Speculation

E' stata introdotta una nuova classe di istruzioni definite *speculative load*, che possono essere spostate prima di una o più ramificazioni. Dove prima il codice prevedeva una normale load viene inserita un'istruzione di *check*. Se l'istruzione eseguita speculativamente solleva un'eccezione, questa viene differita e nel registro target viene scritto un valore NaT (Not a Thing – nei registri GR viene settato il NaT−bit, in quelli FP esiste un vero e proprio valore NaT, simile ai NaN e Inf dei formati floating point dell'IEEE). Poiché le istruzioni speculative propagano le eccezioni differite copiando lo stato NaT, è sufficiente una sola istruzione check per verificare la riuscita di una catena di operazioni speculate. Se il check trova un valore NaT richiama una procedura di recovery (creata dal compilatore) che riesegue l'operazione in maniera *non* speculativa.

Data Speculation

Quando staticamente non è possibile determinare su un'operazione di *store* e una di *load* accedono a regioni di memoria sovrapposte, si ha una dipendenza di memoria ambigua e non è possibile schedulare le istruzioni in un diverso ordine. Per superare questa limitazione sono state introdotte le istruzioni di *advanced load*. Simmetricamente a quanto accade nella control speculation è possibile anticipare una lettura prima di un gruppo di scritture e sostituirla con una *check load* o un *advanced load check*, che in caso di necessità provvedono a ricaricare il valore dalla memoria, rispettivamente in modo diretto o saltando ad una procedura di recovery.