abstract

* scalar architecture processosr為SISD processor
* 提出在scalar architecture中對VLIW做DBT的方法
* 提出一個考慮VLIW特性的高等模擬演算法

Introduction

* multiprocessor systems on chip (MPSoC) is trend
* MPSoC適合處理網路或多媒體
* 使用VLIW exploiting at compile time, and so with high energy efficiency, instruction level parallelism
* system simulation需要有效率的做instruction set interpretation
* 在現行的MPSoC simulation，DBT是很有效率的
* deal with the general problem of dynamically translating code that features explicit ILP, statically defined non unit latency register updates and non immediate control flow changes.
* 兩個主要的貢獻
  + 提出了在VLIW架構下的scalar processor 的instruction interpretation algorighm
  + 提出對DBT更精確的algorithm及其實做

Related works

* DBT, SBT;
* IR is machine independent, and it can be viewed as a generic machine instruction set

Background

* VLIW
  + 需要自行找出ILP
  + n-issue VLIW : 在同一個cycle執行n條獨立指令
  + A set of parallel instructions is called an execute packet (EP).
  + VLIW可以不做stall或bypass就解決hazards
  + compiler需要小心控管每道指令的latency
  + 以c6x為例，delay slot很多
* DBT
  + 從scalar轉到VLIW時，因為VLIW會使用register renaming來找出ILP，這一點會讓translation time縮短
  + 會將target code切成多個block做轉換，並將轉換後的block cache起來，往後如果遇到相同的block就不再做轉換

VLIW Architecture Simulation With DBT

* 一般的DBT對VLIW來說有三個問題
  + WAR的問題，因為指令間latency有差，為了利用因為latency差異所產生的delay slot，會在latency長的指令(A)後面塞指令，當這些指令用到了A的目的暫存器，DBT會認為有hazard要等到A執行完，但實際上這些指令是要用A未完成前的舊資料
  + DBT沒有考慮latency的問題，可能導致運算結果出錯
  + 沒有考慮到delay slot，可能造成delay slot的指令沒有做轉換
* Algorithm for VLIW Architecture Simulation
  + 可以通用到多種不同的simuators上
  + The main idea is to use multiple memory locations for the same target register.
  + simulator在讀data進register時必須要知道它使用到的replicate是哪個
  + register replicate產生的演算法，有實例
* Algorithm Specialization for DBT
  + Efficient Replicates Allocation
    - 假設一個EP中latency最久的指令它的latency為p，因為在同一個時間點，最多只會有p個replicate及register現在的值，因此可以找出一個registers的replicate總數的upper bound
    - 假設一共有n個target registers，那可以事先產生一個長度為n\*(p+1)的circular buffer，之後simulator就可以將這個buffer當作operand去操作replicates
    - 可以不用每遇到register做write才去產生replicate，明顯的空間換時間
  + Problems Induced by TB Granularity
    - However, the TB translation granularity imposed by DBT brings up new problems.
    - Canonical state between TBs

為了要讓不同的TB(translation block)間的起始replicate位置相同，在每個TB的最後面，根據TB中有做寫入的register，將它最後使用的replicate寫回0號位置

* + - Necessary context saves at execution time

canonical的規則是在TB結束時做初始化，但有可能TB中的指令還沒做完，因此它的replicate還沒寫入，會導致之後的TB使用錯的值

提出dynamic waiting queue

This mechanism modifies the simulated processor state as the information must be alive between TBs during execution.

當執行到TB的結尾時如果還有指令待在static waiting queue，就呼叫一個context save去紀錄這個指令的一些資訊，function的參數為指令要寫入的register ID, 要寫入的值所在的replicate, 及還要再等待的cycle數

* + - Context restore

當context save的cycle數到了時，要再呼叫另一個fuction: context restore 去寫入replicate，因此在每個EP結尾都要呼叫一次確保replicate有寫入

* + - Branch instructions special case

當TB結尾有branch進dynamic queue時，那在下一個TB執行時會做到一半就要branch

?????多此一舉?

* + Predicated Instructions Handling
    - 當branch跳或不跳時，後續指令使用的replicate會不同，造成轉譯上的困難
    - 列出四個branch情況一一解決
    - 當branch指令的latency為0時

當branch成立就正常執行，不成立時則將舊的replicate的值複製一分到新的replicate上

* + - 當branch指令的latency不為0時

用一組變數cf\_x去紀錄branch跳或不跳，之後在根據cf的值去決定要不要copy到新的replicate上

* + - if else寫入同一個register

translator可以找出這類關係，前一個指令分配replicate，後一個指令沿用這個replicate

* + - 123