Speculative Precomputation

by Carlos Cunha

University of Minho

Portugal

Outline

- Introduction
- Speculative Precomputation
- SP Tasks
  - Delinquent Loads identification
  - P-Slices
  - Triggers establishment
- Software based SP (SSP)
- SSP implementation on Itanium
- Conclusion

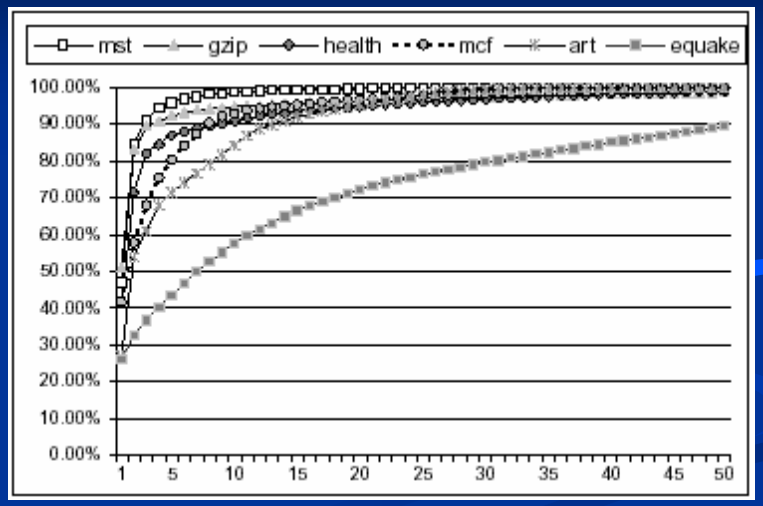
Introduction

- The difference between the speed of computation and the speed of memory access continues to grow (CPU-memory gap)
- The actual processors have Simultaneous Multithreading (SMT) characteristics, improving overall throughput under a multiprogramming workload
- The use of various thread contexts on SMT, only improve the system performance when there are more than one thread executing

Speculative Precomputation (SP)

- SP uses idle hardware thread contexts to execute speculative threads that attempt to avoid future cache misses
- Data are loaded in advance from memory and passed to the non-speculative thread
- SP targets static loads that cause most stalls in the non-speculative thread.

Speculative Precomputation (SP)



SP - Tasks

- Speculative Precomputation follows a set of mandatory steps:
  - identification of delinquent loads
  - construction of p-slices for these loads
  - establishment of triggers.
- This work can be performed with compiler assistance as well as some hardware support

SP – Delinquent Loads

- For most programs, the cache misses are dominated by a few static loads
- We call these poorly behaved loads, delinquent loads
- Identification of delinquent loads is determined by memory access profiling, performed by a compiler or a memory access simulator

SP – P-Slices

- Once a load is identified as delinquent, a p-slice is constructed to prefetch the load
- P-slices are spawned when encountering a trigger, which occurs when a designated trigger instruction in the main thread reaches a particular stage in the pipeline

P-Slices - Example

```
struct DATATYPE {
  int val[10];
};

DATATYPE * data [100];

for(j = 0; j < 10; j++) {
  for(i = 0; i < 100; i++) {
    data[i]->val[j]++;
  }
}
```

```
        loop:
I1  load    r1=[r2]
I2  add     r3=r3+1
I3  add     r6=r3-100
I4  add     r2=r2+8
I5  add     r1=r4+r1
I6  load    r5=[r1]
I7  add     r5=r5+1
I8  store   [r1]=
I9  blt     r6, l
```

Constructed p-slice

```
load   r1=[r2]
add    r1=r4+r1
load   r5=[r1]

Live-in Set
r2, r4

Delinquent load
(instruction I6)
is trigger
```


Software based SP (SSP)

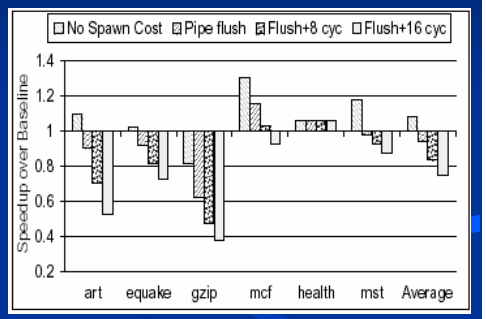
- SSP requires two basic mechanisms to support thread spawning:
  - a mechanism to bind a spawned thread to a free hardware context
  - a mechanism to transfer necessary live-in values to the child thread
- A machine that employs ideal, one-cycle flash copy between registers files of two thread contexts, permits the non-speculative thread to spawn speculative threads instantly

SSP – Itanium Implementation

- An ideal machine may be difficult to implement on processors like the Itanium family processors, due to the cost of implementing a flash copy mechanism for such large register files.
- In the case of Itanium, we can explore a less aggressive but more practical software-based SP (SSP) approach
- The Itanium processor family has:
  - on-chip memory buffers, which are used as spill area for the backing store of the Register Stack Engine (RSE), called Live-in Buffer (LIB)
  - the Lightweight Exception Recovery mechanism (LER), which is used to recover from incorrect control and data speculations.

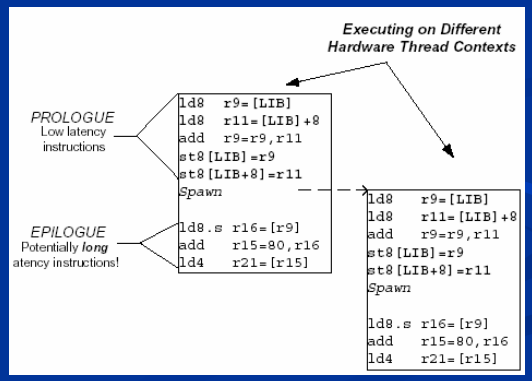
SSP – Itanium Implementation

- Spawning a thread on the SSP approach is no longer instantaneously as the hardware approach
- The non-speculative thread will slow down by the time necessary to invoke and execute the exception handler, and the p-slices must be modified to first load their values from the LIB.

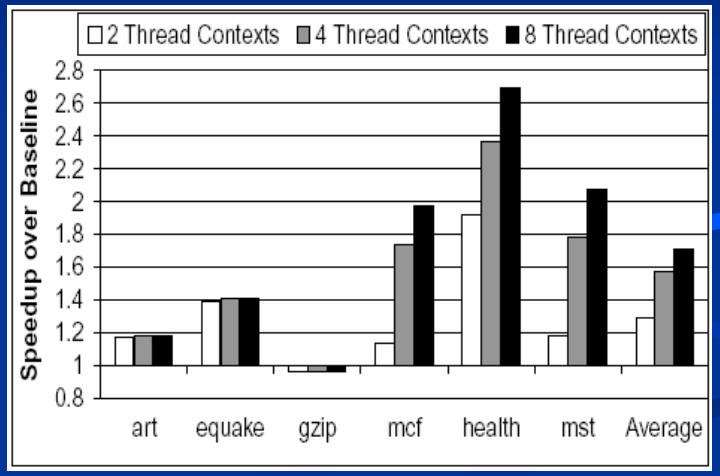


Chaining Triggers

- Chaining p-slices are able to spawn future instances of themselves, decoupling thread spawning from non-speculative thread execution.



Chaining Triggers - Performance



Conclusion

- When spawning threads fall on the main non-speculative thread (via basic triggers), the potential speedup is as high as 30% assuming fast register copies between thread contexts
- However, under more realistic assumptions, the potential speedup is significantly reduced
- When the speculative threads can also spawn other speculative threads (via chaining triggers), these speedups are as high as 169% and average 76% over all benchmarks
- This is achieved via a software based mechanism that can utilize existing Itanium processor features with very little additional hardware support.