Simultaneous Multithreading

Pratyusa Manadhata, Vyas Sekar
{pratyus,vyass}@cs.cmu.edu

1 Introduction:

Current research in processor technology and computer architecture is motivated primarily by the need for greater performance. In this context, it is well understood that the performance gain from improving the memory system alone is limited, and using system Level Integration (such as supporting graphics/sound on chip) can only lead to marginal performance benefits. The most significant gain can be achieved by increasing parallelism in execution.

There exist two kinds of parallelism in typical programming workloads, Instruction Level Parallelism (ILP) and Thread Level Parallelism (TLP). Modern superscalar architectures are designed to capture ILP in programs, while multithreaded and multiprocessor systems are designed to capture TLP or parallelism across threads/processes.

The better solution then would be to exploit both ILP and TLP ; TLP from either multithreaded parallel programs or from multiprogramming workload, and the ILP from each thread.

Neither superscalar nor multiprocessor (MP) can capture ILP and TLP in its entirety and these are inherently incapable of adapting to dynamic levels of ILP and TLP.

This is the primary motivation for a new architecture of processors called Simultaneous Multithreading (SMT).

2 SMT

In this section we identify some of the key characteristics of an SMT architecture and some of the design requirements that can facilitate the implementation of an SMT over a conventional superscalar architecture. The characteristics of SMT processors are
1. inherited from superscalar: issue multiple instructions per cycle
2. from multithreaded: maintain hardware state for multiple threads

In Fig 1 we can see that there is a significant amount of wastage of issue slots in the superscalar and the multithreaded system. There are essentially two kinds of waste: vertical waste (an entire cycle is unused) and horizontal waste (within a cycle issue slots are unused). Superscalar processors look at multiple instructions from same process, and have both horizontal waste (as a result of insuffi-

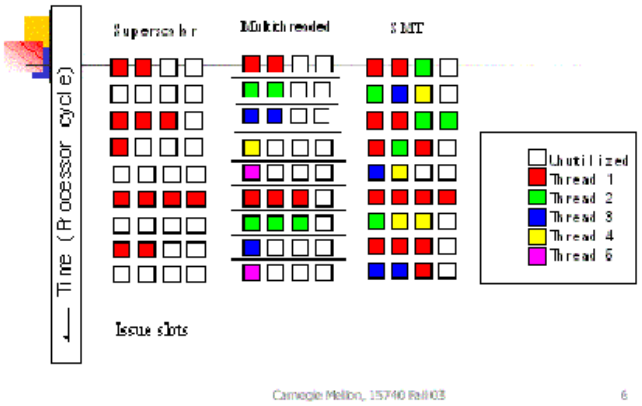

Figure 1: Horizontal and Vertical Wastage

cient ILP) and vertical waste (due to data dependencies and long latency operations). The MT system minimizes vertical waste as it can look at multiple threads to fetch from in each cycle and thus it can tolerate long latency operations within each thread.

3 SMT Model

1. Consider a superscalar that fetches 8 instructions from the IC

2. SMT h/w modifications required over a conventional superscalar
   1. state for h/w contexts for threads
   2. per-thread exception/retirement mechanisms

3. Most h/w resources are available unlike in static resource allocation This implies that a non-parallelizable program will still run efficiently in SMT.

4. Fetch Mechanism:
   a. 2.8 scheme: select 2 threads . fetch 8 from each thread
   (2.4 scheme?) out of these choose a subset to match h/w decoding b/w b. h/w cost:additional port on IC (2.8 better than 2.4)
   c. icount technique: selecting the thread, higher priority to those threads that have least number of instructions in the decode,rename and queue pipeline stages: even distribution, prevents starvation etc. Other options are misscount, bcount etc.

5. Caveat: Hardware register file is larger: 2-clock latency to access register needs 2-cycle read/write.

3.1 SMT Disadvantages

- There is greater register pressure and greater per thread latency due to the longer pipeline.

- On a multiprogrammed workload there is greater stress on shared structures such as BPB, cache, TLB etc.

- A Parallel Workload tends to stress the functional units more.

4 Results and Observations

It has been observed that superscalars approximately give an IPC of about 1-2. But the results shown indicate that SMT can reach an IPC of upto 6.7 (for a 8-issue architecture). Even though the SMT pipeline is longer implying a longer latency for a single thread it is observed to not have a significant performance effect. The reason for the non-degraded performance in the presence of conflicts and a longer pipeline is essentially the systems' ability to absorbs additional conflicts i.e., the ability to hide latency by using multiple issues from multiple threads. The multiprocessor architectures MP2 and MP4 were observed to be hindered by static resource partitioning, while SMT on the other hand dynamically partitions resources among threads. Also a comparison between MP2 vs MP4 shows that MP2 can better adapt to ILP, while MP4 is better suited for utilizing TLP, which is quite intuitive as there are more functional units per processor available in the MP2, while there are more parallel units in the MP4. SMT can also lead to increased cache misses/conflicts and greater stress on the branching hardware. However the impact on overall program performance is not significant as SMT, efficient hardware design, and compiler optimizations can hide latencies and conflicts significantly. The key insight is that SMT achieves a better performance gain than Superscalar, multithreaded, and multiprocessor architectures due to the ability to ignore the distinction between ILP and TLP which implies that resources are not statically partitioned.

4.1 Discussion of Issues in SMTs

- Cost vs Performance: It is necessary to quantify the architecture that can best use the chip area and can provide enhanced performance with minimal hardware overhead.
- Quantitative Comparisons: It is difficult to quantify in absolute terms the performance gain that the SMT processor can deliver. Often this depends a lot on design cycle time, the actual hardware implementation etc that are hard to predict given the technology trends.
- Compilers: One of the earlier claims was that SMT is easier for compilers and programmers, as the hardware can dynamically repartition resources. But the general feeling is that in order to assure a performance no worse than the competing architectures and to ensure maximum processor utilization, one does need compiler support for identifying sources of parallelism and help in static scheduling.
- OS: It is important to consider OS issues such as thread scheduling, thread priority etc. that will be necessary in a realistic implementation of an SMT, and the interaction between the thread priority and the fetch/issue logic is an interesting issue.

- Another observation is that more than static partitioning of resources in multiprocessors the communication overhead is a significant reason why SMTs perform better than MPs.

- The question also arises whether SMT needs a branch prediction mechanism at all? The answer is yes, which is again consistent with the design philosophy that a non-parallelizable program still needs to get a good performance.

- Is the performance gain adequate with the additional resource cost? It has been shown that an SMT outperforms an equally resource-equipped multiprocessor running at maximum number of supported threads, which shows that the SMT has maximum resource utilization.

What does the future hold for SMTs?
Each processor in an SMP can use SMT - This is a direct extension of the SMP and SMT architectures that can create small to massive parallel systems where each processor employs SMT to minimize execution time. It has been observed that next generation architectures would be based on design issues that tend to maximize use of power and chip area, and this would mean that multiprocessing (MP or MT or SMT) on chip is more efficient than a wider superscalars.
An interesting observation is that even though the research on SMT was done in the mid-late 90s, the actual commercial implementation of an SMT on a processor has been delayed until now (the Intel "Hyperthreading" Pentium). This shows that chip-design in reality is far more complex, and there are other economic factors that come into play.

References

[1] Susan Eggers, Joel Emer, Henry Levy, Jack Lo, Rebecca Stamm, and Dean Tullsen. *Simultaneous Multithreading: A Platform for Next-generation Processors, in IEEE Micro, September/October 1997, pages 12-18.*

[2] Jack Lo, Susan Eggers, Joel Emer, Henry Levy, Rebecca Stamm, and Dean Tullsen. *Converting Thread-Level Parallelism Into Instruction-Level Parallelism via Simultaneous Multithreading, in ACM Transactions on Computer Systems, August 1997, pages 322-354.*

[3] Dean Tullsen, Susan Eggers, Joel Emer, Henry Levy, Jack Lo, and Rebecca Stamm. *Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor , in Proceedings of the 23rd Annual International Symposium on Computer Architecture, May 1996, pages 191-202.*