Simultaneous Multithreading: Maximizing On-Chip Parallelism

文章针对同时多线程技术（一种允许几个独立的线程在一个周期内向一个超标量的多个功能单元发出指令的技术）。

Introduction

同步多线程的目标是在面对长内存延迟和每个线程可用并行性有限的情况下，大幅提高处理器利用率。同时多线程结合了现代拆标量处理器的多问题质量特性和多线程架构的延迟隐藏性。

目前的微处理器采用现代超标量或多线程架构提高性能和处理器利用率。传统的多线程隐藏了内存和功能单元延迟，解决了垂直浪费。同时多线程可以同时解决水平和垂直浪费。

本研究评估了不同的同步多线程模型相对于宽标量架构和传统线程架构的潜在改进。

超标量执行和传统多线程在提高指令吞吐量方面的局限性：

- 即使有8个输出的超标量架构，每个周期也无法维持1.5条指令
- 细粒度多线程处理器，无论线程数量如何，都只能利用大约40%的宽标量

同时多线程在指令吞吐量方面提供了显著的性能改进，并不受处理器带宽的限制

本文工作在多个方面有扩展

- 方法论，包括我们模拟的准确性和系节，用于比较的基础架构、编译优化和调度技术
- 模拟的各种SM模型
- 多同步多线程缓存交互的分析
- 对多处理和同时多线程的比较和评估

Methodolgy

模拟环境

模拟器支持部分解码指令的缓存，对执行流水线、内存层次结构（命中率和带宽方面）、TLB和宽超标量处理器的分支预测逻辑进行了建模

假设第一级和第二级芯片上的缓存要比Alpha上的大

- 多线程对缓存子系统施加了更大的压力
- 希望在同时多线程可行的同一时间空间内有更大的缓存

	ICache	DCache	L2 Cache	L3 Cache
Size	64 KB	64 KB	256 KB	2 MB
Assoc	DM	DM	4-way	DM
Line Size	32	32	32	32
Banks	8	8	4	1
Transfer time/bank	1 cycle	1 cycle	2 cycles	2 cycles

Table 2: **Details of the cache hierarchy**

模拟器支持有限的动态执行，使用多流跟踪调度编译器实现直接输出模型。减少了完全动态执行可能带来的好处。

一个2048条目直接映射的2位分支预测历史表支持分支预测，提高了分支地址的覆盖范围。

不需要对基本流水线做任何改变来适应同步多线程。

H3 工作负载

工作负载时SPEC92基准测试套件，模型通过多路编程而不是并行处理实现的并行工作负载。这样，吞吐量结果不会收到同步延迟、低效的并行处理等影响。

H2 Superscalar Bottlenecks: Where Have All the Cycles Gone?

实验表明不存在浪费输出带宽的主要来源，尽管在单个应用程序中有一些主要的项目，但在每种情况下，主要原因是不同的。指令调度的目标是浪费输出贷款的几个重要部分。

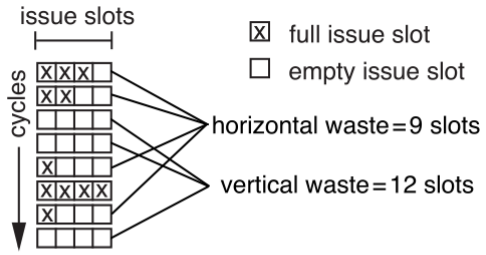

Figure 1: **Empty issue slots can be defined as either vertical waste or horizontal waste. Vertical waste is introduced when the processor issues no instructions in a cycle, horizontal waste when not all issue slots can be filled in a cycle. Superscalar execution (as opposed to single-issue execution) both introduces horizontal waste and increases the amount of vertical waste.**

61%是垂直浪费，其余是水平浪费。

H2 Simultaneous Multithreading

H3 The Machine Models

- 细粒度多线程

  每个周期只有一个线程发出指令，但是它可以使用处理器的整个发出宽带。

- SM：完全同步输出

  所有8个线程在每个循环中竞争每个输出槽。就硬件复杂度而言，这是最不现实的模型，但它提供了对并发多线程的潜在可能性的洞察

- SM：单发射，双发射，四发射

  这三种模型限制了每个线程可以发射的指令数量，或者每个周期调度窗口中活动的质量数量

- SM：有限链接

  每个硬件上下文都直接连接到每种类型的功能单元中的一个

Model	Register Ports	Inter-inst Dependence Checking	Forwarding Logic	Instruction Scheduling onto FUs	Notes
Fine-Grain	H	H	H/L*	L	Scheduling independent of other threads.
SM:Single Issue	L	None	H	H	
SM:Dual Issue	M	L	H	H	
SM:Four Issue	M	M	H	H	
SM:Limited Connection	M	M	M	M	No forwarding between FUs of same type; scheduling is independent of other FUs
SM:Full Simultaneous Issue	H	H	H	H	Most complex, highest performance

∗ We have modeled this scheme with all forwarding intact, but forwarding could be eliminated, requiring more threads for maximum performance

Table 4: **A comparison of key hardware complexity features of the various models (H=high complexity). We consider the number of ports needed for each register file, the dependence checking for a single thread to issue multiple instructions, the amount of forwarding logic, and the difficulty of scheduling issued instructions onto functional units.**

H3 The Performance of Simultaneous Multithreading

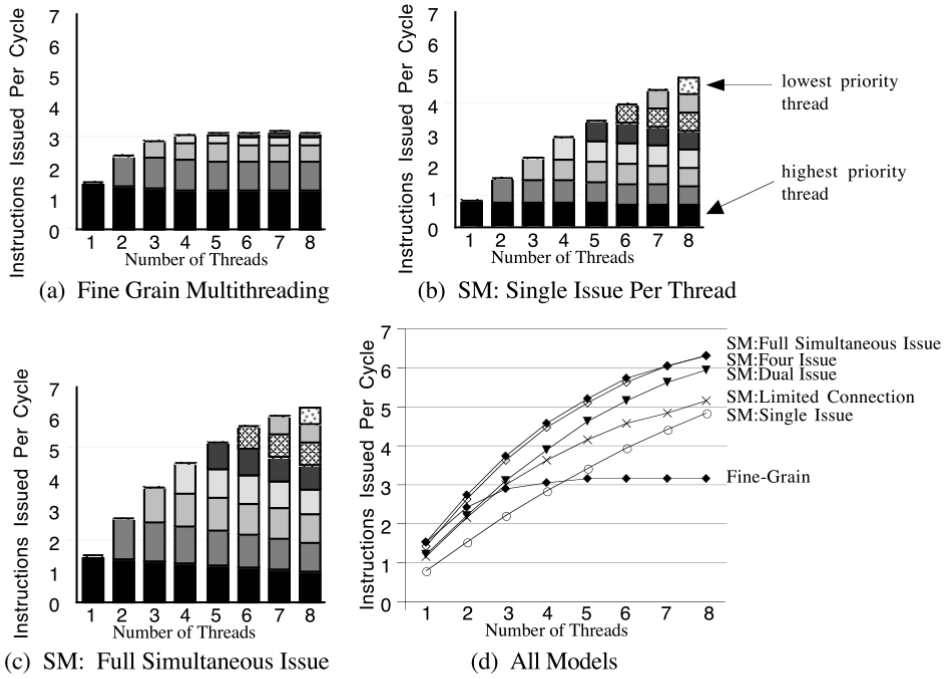

Figure 3: **Instruction throughput as a function of the number of threads. (a)-(c) show the throughput by thread priority for particular models, and (d) shows the total throughput for all threads for each of the six machine models. The lowest segment of each bar is the contribution of the highest priority thread to the total throughput.**

细粒度多线程架构提供了最大的加速，比单线程执行提高2.1。且此模型增加4个以上的线程并没有太大变化。

处理器利用率的增加是线程动态共享处理器资源的直接结果，否则这些资源会在大部分时间内保持空闲。缓存共享会降低性能

在宽标量上运行时，并发多线程超过了单线程执行或细粒度多线程所能达到的性能限制。

H2 Cache Design for a Simultaneous Multi-threaded Processor

本文假设所有线程共享二级和三级缓存，只比较一级缓存的共享性和数据缓存的使用。

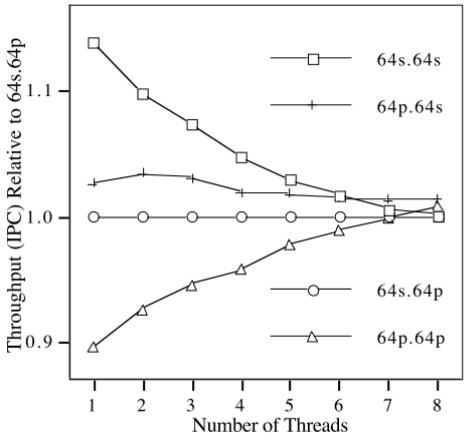

Figure 4: **Results for the simulated cache configurations, shown relative to the throughput (instructions per cycle) of the 64s.64p cache results.**

共享缓存针对少量线程进行了优化，而私有缓存在大量线程时性能更好。

共享数据源缓存在数据共享环境中有一个显著的优势，它允许在数据缓存层次结构的最底层进行共享，而不需要任何特殊的硬件来实现缓存一致性

SimultaneousMultithreadingversus Single-Chip Multiprocessing

随着芯片密度的不断提高，单芯片多处理器将提供一种明显的方式来实现并行。

实验中保持以下全部或大部分相等：寄存器数量、总的输出带宽、特定的功能单元配置（针对多处理器进行优化的，代表了同时多线程的低效配置）

Purpose of Test	Common Elements	Specific Configuration	Throughput (instructions/cycle)
Unlimited FUs: equal total issue bandwidth, equal number of register sets (processors or threads)	**Test A**: FUs = 32 Issue bw = 8 Reg sets = 8 **Test B**: FUs = 16 Issue bw = 4 Reg sets = 4 **Test C**: FUs = 16 Issue bw = 8 Reg sets = 4	SM: 8 thread, 8-issue	6.64
		MP: 8 1-issue procs	5.13
		SM: 4 thread, 4-issue	3.40
		MP: 4 1-issue procs	2.77
		SM: 4 thread, 8-issue	4.15
		MP: 4 2-issue procs	3.44
Unlimited FUs: Test A, but limit SM to 10 FUs	**Test D**: Issue bw = 8 Reg sets = 8	SM: 8 thread, 8 issue, 10 FU	6.36
		MP: 8 1-issue procs, 32 FU	5.13
Unequal Issue BW: MP has up to four times the total issue bandwidth	**Test E**: FUs = 32 Reg sets = 8 **Test F**: FUs = 16 Reg sets = 4	SM: 8 thread, 8-issue	6.64
		MP: 8 4-issue procs	6.35
		SM: 4 thread, 8-issue	4.15
		MP: 4 4-issue procs	3.72
FU Utilization: equal FUs, equal issue bw, unequal reg sets	**Test G**: FUs = 8 Issue BW = 8	SM: 8 thread, 8-issue	5.30
		MP: 2 4-issue procs	1.94

Figure 5: **Results for the various multiprocessor vs. simultaneous multithreading comparisons. The multiprocessor always has one functional unit of each type per processor. In most cases the SM processor has the same total number of each FU type as the MP.**

在各种配置下，由于功能单元的动态分区，同时多线程优于单芯片多处理。SM需要更少的资源来实现给定的性能水平。

其他优势

- 达到性能需要更少的线程

  其他处理器可以扩展使用某空间处理器留下的空间资源

  - 运行并行代码，其中并行度随时间变化
  - 在目标环境中少量线程的性能很重要
  - 工作负载的大小与机器的大小相同
- 设计的力度和灵活性

总结

与其他架构相比，同时多线程的优点有

- 一个同步的多线程架构，经过适当的配置，在相同的问题宽度下，可以实现4倍的指令
- 同时多线程比细粒度多线程性能好2倍
- 在给定相同的寄存器集和功能单元综述的情况下，并发多线程体系结构在性能上优于多发射多处理器。

与其他方法相比，同时多线程的优点是它能够通过在多个线程之间动态调度功能单元来提高利用率。SM还增加了硬件设计的灵活性;同时多线程体系结构可以权衡功能单元、寄存器集和发布带宽以获得更好的性能，并且可以以细粒度的方式添加资源。与超级标量相比，多线程同时增加了指令调度的复杂性，并导致共享资源争用，特别是在内存子系统中。