CS533: Chip Multiprocessors

Josep Torrellas

University of Illinois in Urbana-Champaign

Introduction

- Increasing chip density
  - Over 100 billion transistors/chip already
- Conventional dynamic-issue superscalar approach
  - ILP limited in single thread of control
  - Large issue window $\rightarrow$ more complexity, less returns
- Optimal usage of silicon space needed
  - Simultaneous multithreading (SMT)
  - Chip multiprocessor (CMP)

SMT Motivation

- Wide superscalar is the way to go
  - Must continue to improve single-program performance
- Multiple programs share fetch & issue bandwidth
  - Thread-level parallelism (TLP) improves throughput of wide superscalar
  - Can still exploit high ILP in a single program

CMP Motivation

K. Olukotun et al. "The case for a single-chip multiprocessor" *ASPLOS*, 1996

- Wide superscalar is not the way to go
  - Trades fast clock for minor IPC gain
  - Design and verification complexity
- Multipe simple processors (e.g., 2-way or 4-way superscalar)
  - Exploit TLP
  - Moderate ILP plus fast clock

Aggressive Superscalars

- Multiple instruction issue, dynamic scheduling, speculative execution, non-blocking caches, ...
- Trend
  - Wider instruction issue
  - Larger amounts of speculative execution
- However,
  - Limited amounts of ILP
  - Fundamental circuit limitations

$\Longrightarrow$ Better resource utilization: Multiple processors on a chip

Dynamic Superscalar CPU

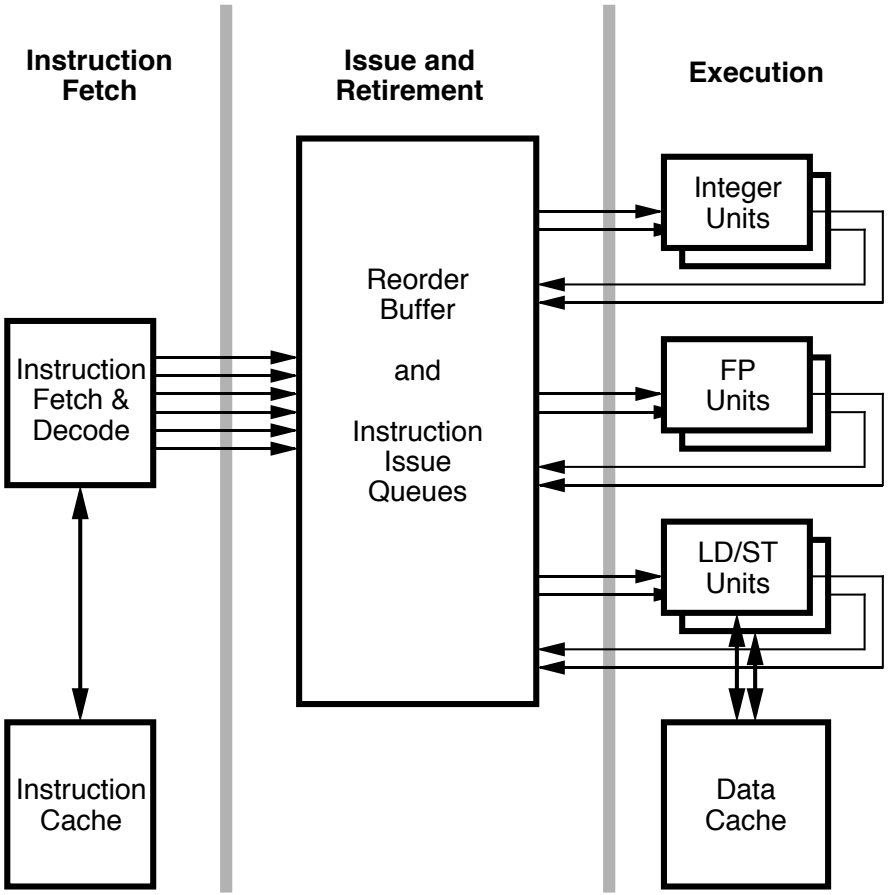

Figure 1. A dynamic superscalar CPU

tures called reservation stations [3]. It is possible to design a dynamically scheduled superscalar microprocessor using reservation stations; Johnson gives a thorough description of this approach

achieving this. A technique that divides the instruction cache into banks and fetches from multiple banks at once is not too expensive to implement and provides performance that is within 3% of a perfect scheme on an 8-wide issue machine. Even with good branch prediction and alignment a significant cache miss rate will limit the ability of the fetcher to maintain an adequate window of instructions. There are still some applications such as large logic simulations, transaction processing and the OS kernel that have significant instruction cache miss rates even with fairly large 64 KB two-way set-associative caches [19]. Fortunately, it is possible to hide some of the instruction cache miss latency in a dynamically scheduled processor by executing instructions that are already in the instruction window. Rosenblum *et. al.* have shown that over 60% of the instruction cache miss latency can be hidden on a database benchmark with a 64KB two way set associative instruction cache [19]. Given good branch prediction and instruction alignment it is likely that the fetch phase of a wide-issue dynamic superscalar processor will not limit performance.

- Instruction queue often implemented as multiple instruction queues for different type of instructions
- Stages:
  - Fetch
  - Issue
  - Execute

In the issue phase, a packet of renamed instructions is inserted into the instruction issue queue. An instruction is issued for execution once all of its operands are ready. There are two ways to implement renaming. One could use an explicit table for mapping architectural registers to physical registers, this scheme is used in the R10000 [24], or one could use a combination reorder buffer/instruction queue as in the PA-8000 [14]. The advantage of the mapping table is that no comparisons are required for register renaming. The disadvantage of the mapping table is that the number of access ports

Fetch Phase

- Goal: Present a large window of decoded instructions
- Constraints:
  1. Mispredicted branches
  2. Instruction misalignment
  3. Cache misses

Fetch Phase (continued)

1. Mispredicted branches
   - Branch predictor buffers (e.g., 64 kbits)
     - Selective branch predictor
     - Reduce misprediction $< 5\%$
2. Instruction misalignment
   - Necessary to align a packet of instructions for decoder
   - If issue width > 4, then with high probability, need fetch across a branch for a single packet of instructions
     - Need fetch from 2 cache lines and merge
     - Scheme: Divide the instruction ncache into banks and fetch from multiple banks

Fetch Phase (continued)

3. Cache misses
   - High miss rate $\rightarrow$ limits ability to maintain large window
   - Can hide some cache miss latency by executing other instructions already in the window (dynamically scheduled CPUs)

$\Longrightarrow$ Overall: Fetch not limit

Issue Phase

- Packet of renamed instructions is inserted into instruction issue queue
- An instruction is issued when all operands are ready
- Renaming implemented with RAT, instruction queue and reorder buffer

Issue Phase (continued)

- Once instruction in instruction queue
  - Instructions that issue must update their dependences
  - Many comparators (e.g., HP-PA8000: 20% die area)
- Also: large window to find independent instructions
  - Size of instruction issue queue is large
  - Need broadcast of tags $\rightarrow$ **Wires are slow**

$\Longrightarrow$ Overall: Instruction issue queue wil limit cycle time

Execution Phase

- Operand values: fetched from register file or bypassed from earlier instructions
- Wide superscalar has probelms with
  - Register File
    - Larger to accommodate more rename registers
    - Many ports
    - Complexity $\approx$ #ports^2 $\approx$ issue width^2
  - Bypass logic
    - Complexity $\approx$ #execution units^2
    - **Wire delay**
  - Functional Units
    - More ports needed to data cache

Single-Chip Multiprocessor

- Technology push
  - Benefits of wide issue are limited
  - Decentralized microarchitecture: easier to build several simple fast processors than one complex processor
- Application pull
  - Applications exhibit parallelism at different granularities
  - $<$ 10 instructions/cycle (INT applications)
  - $>$ 40 instructions/cycle (loops in FP applications)

CMP

- INT applications
  - Use moderate issue processor (e.g., 2-way or 4-way) with very high clock rate
- FP applications
  - need compiler to parallelize
  - If cannot parallelize or serial section, CMP runs slow

Microarchitectures Compared

- SS: 6-way superscalar (@ 500MHz)
- CMP: Four 2-way superscalars (@ 500 MHz)

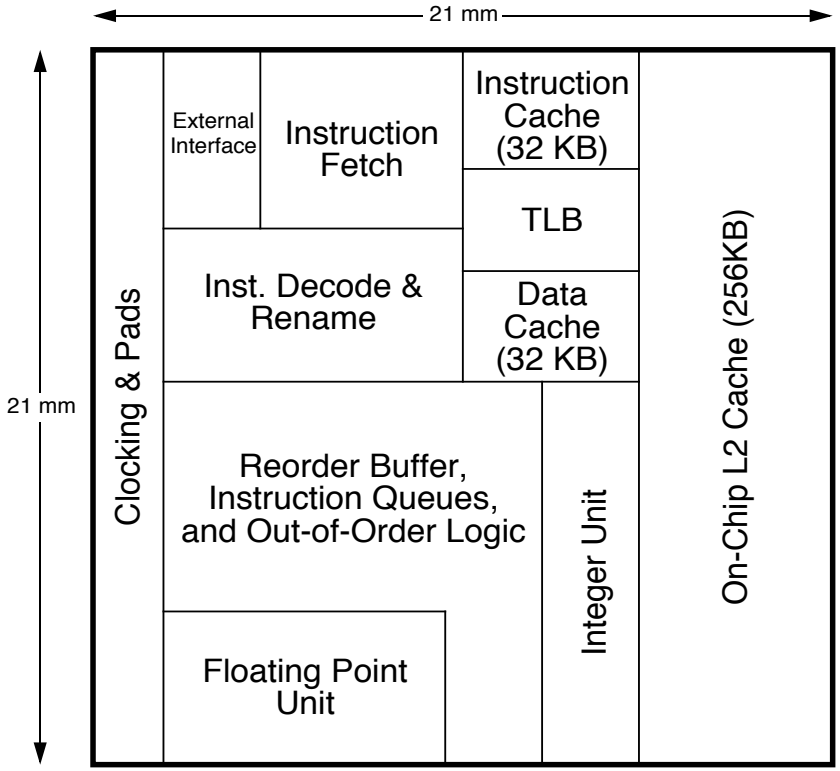

Figure 2. Floorplan for the six-issue dynamic superscalar microprocessor.

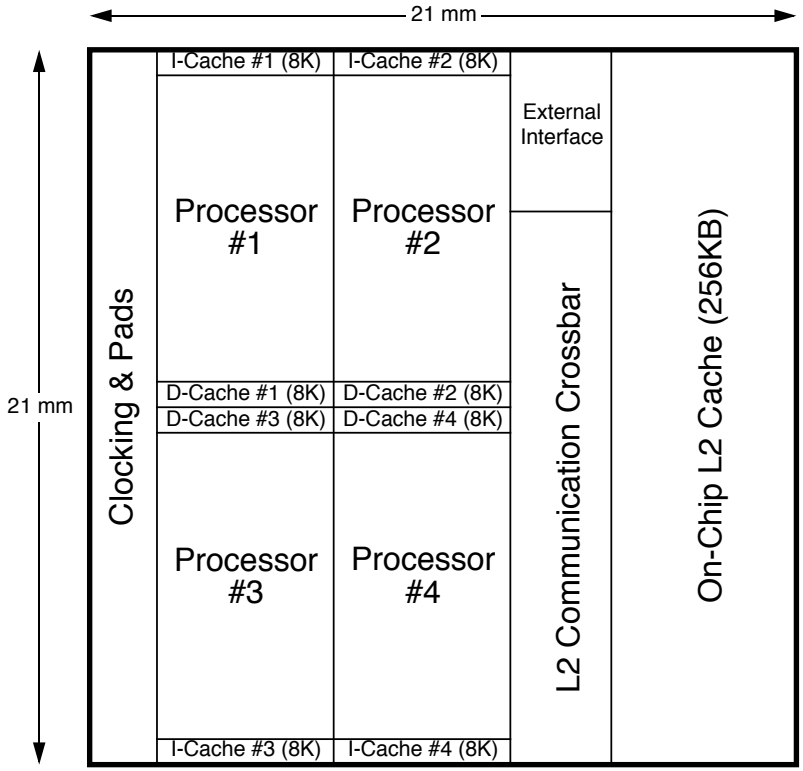

Figure 3. Floorplan for the four-way single-chip multiprocessor.

vents the instruction fetch mechanism from becoming a bottleneck since the 6-way execution engine requires a much higher instruc-

sors is less than one-fourth the size of the 6-way SS processor, as shown in Table 3. The number of execution units actually increases

Performance Comparison

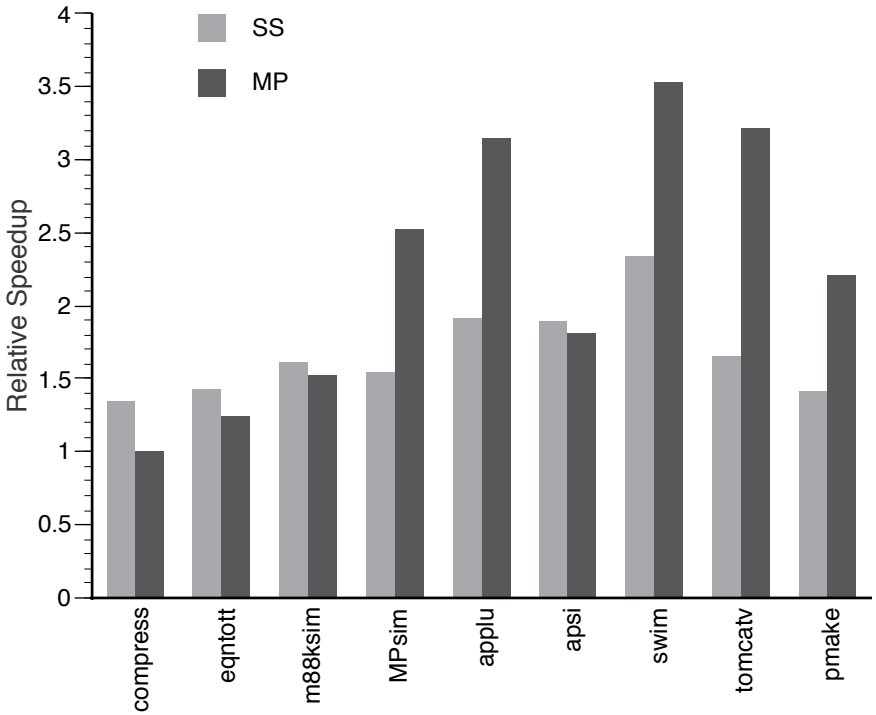

Figure 6. Performance comparison of SS and MP.

Our results show that on applications that cannot be parallelized the superscalar microarchitecture performs 30% better than one processor of the multiprocessor architecture. On applications with fine grained thread-level parallelism the multiprocessor microarchitecture can exploit this parallelism so that the superscalar microarchi-

- ILP only
  - SS is 30% better than CMP
- ILP & fine grain threads
  - SS and CMP comparable
- ILP & coarse grain threads
  - CMP is 1.5-2$\times$ better