HY425 Lecture 08: Limits of ILP

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

October 31, 2011

ILP techniques

Hardware

- Dynamic scheduling with scoreboard
- Dynamic scheduling with renaming
  - Tomasulo, renaming registers
- Branch prediction
- Multiple issue
- Speculation

Software

- Instruction scheduling
- Code transformations (topic of next lecture)

What limits ILP

Software and hardware issues

- Limits of parallelism in programs
    - Data flow – true data dependencies
    - Control flow – control dependencies
    - Code generation, scheduling by compiler
- Hardware complexity
    - Large storage structures – branch prediction, ROB, window
    - Complex logic – dependence control, associative searches
    - Higher bandwidth – multiple issue, multiple outstanding instructions
    - Long latencies – memory system (caches, DRAM)

What is the upper bound of ILP in programs?

Roofline model of performance analysis

- Study of maximum ILP in programs
  - Difficult question dependent on hardware and compiler technology
  - Different conclusions with different assumptions
- Optimistic (unrealistic) assumptions of hardware
  - Unlimited storage resources for tables, etc.
  - Perfect branch prediction and speculation
  - Others . . .

David Wall (DEC, WRL Technical Report, 1993)

Assumptions

- Infinite virtual registers for renaming available
- Branch prediction is perfect
- All branch targets are perfectly predicted
    - No control dependencies
- All memory address known
    - Can move loads before prior to unrelated stores
    - No dependencies other than true data dependencies

David Wall (DEC, WRL Technical Report, 1993)

Assumptions

- Unlimited instruction window size
- Unlimited number of functional units
- All functional units compute in one cycle
- Perfect caches

David Wall (DEC, WRL Technical Report, 1993)

Comparison to a realistic processor – Alpha 21264

- Four-way instruction issue
- 80 renaming registers
- Branch predictor:
    - 1024-branch history, 2 × 8K branch patterns

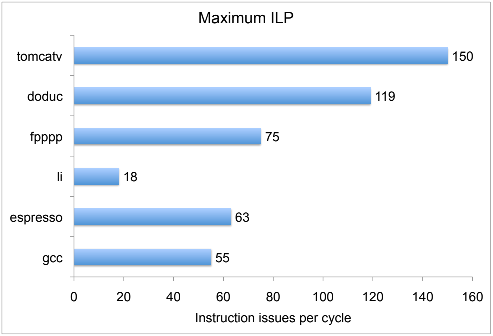
Methodology

- Collect trace of instructions and memory references
- Schedule each instruction “by hand” as early as possible
    - Wait until data dependence resolved

David Wall (DEC, WRL Technical Report, 1993)

Theoretical maximum ILP

- gcc, espresso, li integer programs
- fpppp, doduc, tomcatv floating point programs



Limiting the instruction window size

Perfect processor

- Can look arbitrarily far ahead to fetch instructions
- Can rename output registers for all instructions that can issue
- Can determine data dependencies for all instructions
  - $O(n^2)$ for n instructions
- Provide functional units for all instructions

Limiting the instruction window size

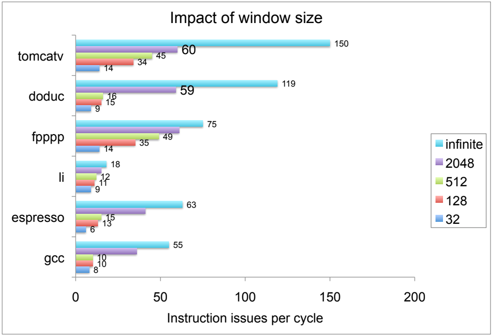
Instruction window

- Group of instructions examined for simultaneous execution
- $WS \times IW \times RPI$ comparators needed
  - WS: window size
  - IW: issue width
  - RPI: registers per instruction to check

Limiting the instruction window size

Window size 32 – ∞

- 32–128 realistic values for modern processors



Realistic branch predictor

Tournament predictor

- 2-bit correlating, 2-bit non-correlating, 2-bit selectors
- 8192 branches, 2 predictors, 1 selector per branch
- Correlating predictor indexed with PC XORed with history
- Non-correlating predictor indexed with PC
- Average accuracy 97% in SPEC

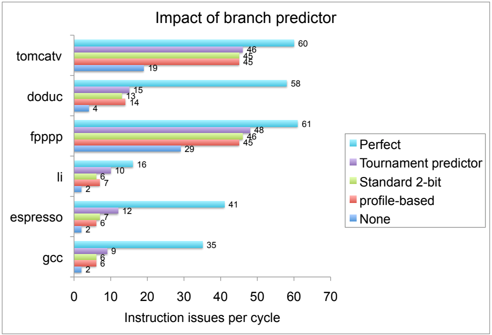
Alternatives

- 2-bit predictor with 512 entries, 16-entry return address table
- Static predictor using profile of application
- No prediction

Realistic branch predictor

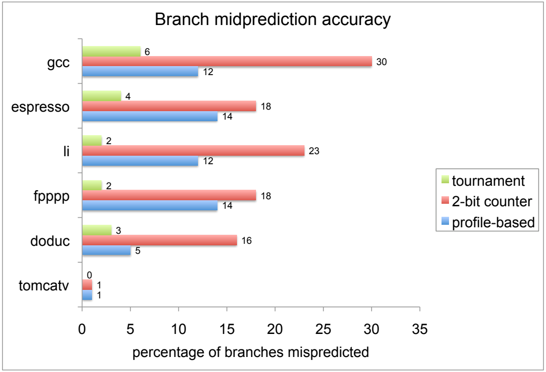
Impact of static vs. dynamic prediction

- 2048-instruction window, 64-way issue, 0-cycle mispredicted branch penalty



Realistic branch predictor

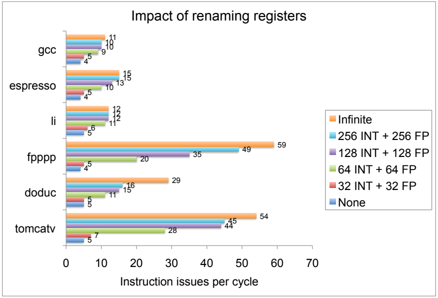
Impact on INT versus FP programs



Number of renaming registers

32 – ∞ renaming registers

- 2048-instruction window, 64-way issue
- 8K-entry tournament predictor



Alias analysis

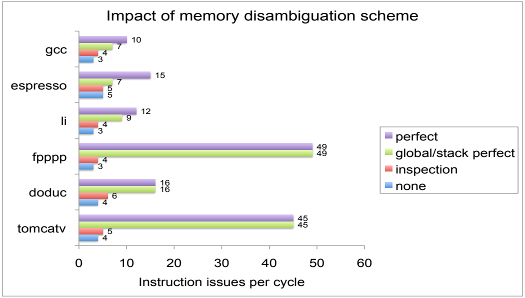
Alternatives for static and dynamic alias analysis

- Impossible to disambiguate all references at compile time
  - Compiler can inspect static data in global segment and stacks (known locations)
  - Hard to inspect data in heap (dynamic allocation, pointers)
- Unbounded number of comparisons needed at runtime
- Three options
  - Perfect disambiguation of global and stack data (perfect compiler)
  - Inspection (disambiguate based on registers pointing to memory)
  - None (no disambiguation)

Impact of alias analysis

Alias alternatives

- 2048-instruction window, 64-way issue, 8K-entry tournament predictor. 256 INT, 256 FP renaming registers



Impact of alias analysis

Implementing memory disambiguation

- Need to know effective addresses of all earlier stores
- Otherwise:
  - In-order address calculation
  - Effective address speculation

Disambiguation with speculation

- Load assumes no dependence or uses dependence predictor
- Stores check for dependence violations upon commit
- Undo and restart mechanism used upon mis-speculation

Going beyond the limits

Advanced hardware techniques for ILP

- Memory WAW and WAR hazards
  - May happen across procedure calls
- Unnecessary dependencies imposed by software
  - E.g. incrementing the loop induction variable
- Predictable data flow
  - Value prediction
    - Prediction of addresses for memory disambiguation
    - Prediction of values

Other considerations for ILP

Clock rate vs. issue width

- 1994 HP PA 7100 @ 99 MHz 2-issue faster than TI SuperSPARC 3-issue @ 60 MHz
- Focus on CPI may trade with long cycle time

Amdahl's Law

- Single improvement may not improve performance
- Resources should scale proportionally

Other considerations for ILP (cont.)

Control flow

- Branches more predictable in FP than INT codes
- FP programs have simpler control paths than INT programs

Parallelism beyond basic blocks

- Multi-program and multi-threaded parallelism
- Limited ILP in a single program motivates simpler using many processors to run many programs
- Multiple simpler processors an attractive alternative for servers

Other considerations for ILP (cont.)

Clock speed

- Increased wire delays prevent increasing clock speed
- Pipeline deepening may:
  - Enable higher clock frequency
  - Increase stall cycles and demand for ILP
  - Require multiple memory accesses, branch predictions, register file accesses per cycle
  - Challenging at GHz clock rates

Other considerations for ILP (cont.)

Power issues

- Increasing complexity increases power consumption
  - More transistors switching
- Increasing complexity and frequency will increase power exponentially
  - More transistors switching at a higher clock rate
- Increasing complexity linearly does not increase performance linearly
  - 3-issue Intel Pentium barely gets CPI < 1.0
  - More switching transistors per unit of performance

Instruction fetch and decode

CISC translated to RISC

- IA-32 CISC instruction set decoded to microops (uops)
- uops scheduled in dynamically scheduled speculative pipeline
- Trace cache:
  - Frequently executed instruction sequences, including non-adjacent sequences
  - Sequences including multiple branches
- Up to 6 uops (three IA-32 instructions) decoded and translated per cycle
- Low miss rate (0.15% for SPEC CPUINT2000)

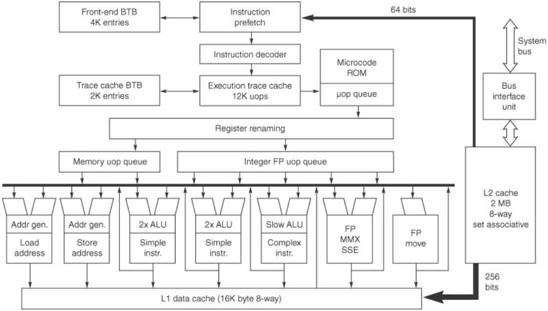
Speculative pipeline

Dynamic scheduling

- Out-of-order execution pipeline
- Register renaming for up to 3 uops per cycle
- Commit up to 3 uops per cycle
- Six dispatch ports to functional units
    - Frequently executed instruction sequences, including non-adjacent sequences
    - Sequences including multiple branches

Microarchitecture of Pentium 4

Pentium 4 640



© 2007 Elsevier, Inc. All rights reserved.

Pentium 4 640 implementation

Microarchitecture quantitative characteristics

- Deep pipelines (21+ stages in various Pentium 4 generations)
- Minimum 31 cycles from fetch to commit

Feature	Size	Comments
Front-end branch target buffer	4K entries	Predicts the next IA-32 instruction to fetch; used only when the execution trace cache misses.
Execution trace cache	12K uops	Trace cache used for uops.
Trace cache branch-target buffer	2K entries	Predicts the next uop.
Registers for renaming	128 total	128 uops can be in execution with up to 48 loads and 32 stores.
Functional units	7 total; 2 simple ALU, complex ALU, load, store, FP move, FP arithmetic	The simple ALU units run at twice the clock rate, accepting up to two simple ALU uops every clock cycle. This allows execution of two dependent ALU operation in a single clock cycle.
L1 data cache	16KB; 8-way associative; 64-byte blocks write-through	Integer load to use latency is 4 cycles; FP load to use latency is 12 cycles; up to 8 outstanding load misses.
L2 cache	2 MB; 8-way associative; 128-byte blocks write back	256 bits to L1, providing 108 GB/sec; 18-cycle access time; 64 bits to memory, capable of 6.4 GB/sec. A miss in L2 does not cause an automatic update of L1.

Performance of Pentium 4

Memory latency

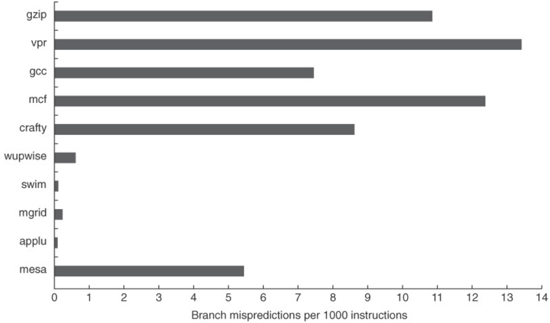
- Performance critically dependent on memory system
- Memory (DRAM) latency upward of 100 cycles
  - Fastest memories as of 2006, 3.2 GHz clock
  - 2 or 3 levels of caches common in modern high-end processors

Branch prediction

- Trace cache and branch prediction
  - Misprediction rate
  - Percentage of instructions misspeculated

Branch prediction on Pentium 4

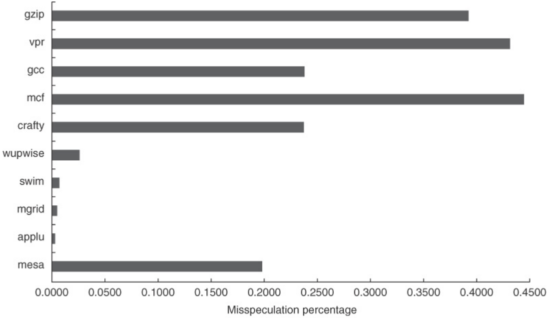
Misprediction rate 8× higher in INT vs. FP programs



© 2007 Elsevier, Inc. All rights reserved.

Branch prediction on Pentium 4

Misspeculated instruction rate follows misprediction rate

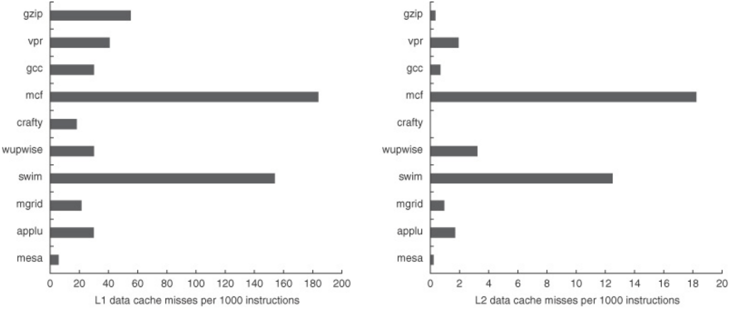


© 2007 Elsevier, Inc. All rights reserved.

Data cache performance on Pentium 4

Multi-level cache hierarchy

- L2 cache miss penalty approx. 10× L1 cache miss penalty
- Hard to overlap stalls due to L2 cache misses

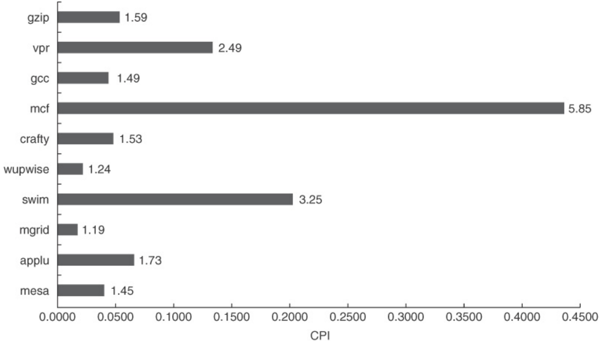


© 2007 Elsevier, Inc. All rights reserved.

CPI on Pentium 4

Is there any ILP to begin with?

- Translation of IA-32 instructions to uops increases CPI by 1.29× (1.29 uops per instruction)

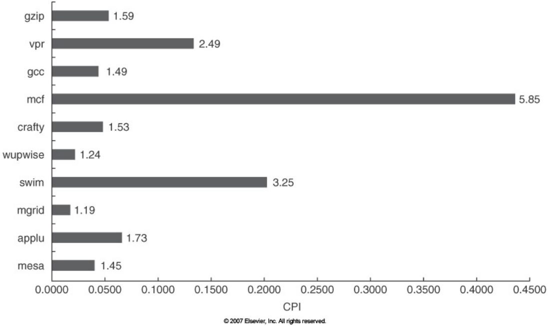


© 2007 Elsevier, Inc. All rights reserved.

CPI on Pentium 4

Is there any ILP to begin with?

- When does the processor get $>$ 1 uop per cycle?

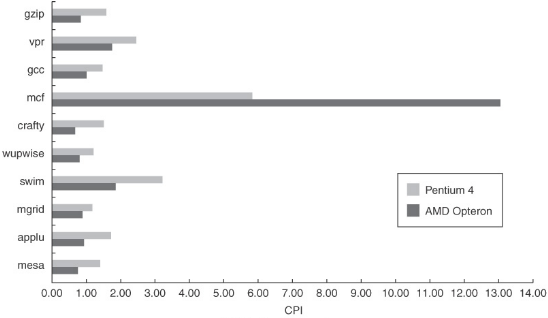


© 2007 Elsevier, Inc. All rights reserved.

Pentium 4 @ 3.2 GHz vs. AMD Opteron @ 2.6 GHz

Putting it all together

- Can a processor with a lower clock frequency outperform a processor with a higher clock frequency?

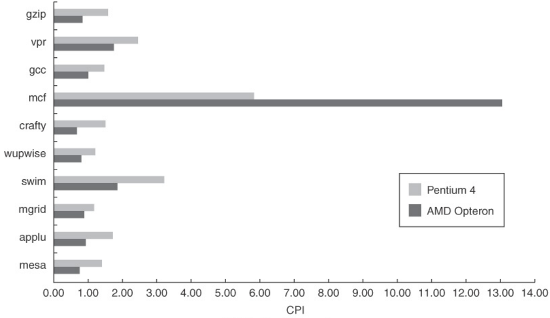


© 2007 Elsevier, Inc. All rights reserved.

Pentium 4 @ 3.2 GHz vs. AMD Opteron @ 2.6 GHz

Putting it all together

- Pentium CPI = 1.27 × AMD CPI
- Pentium clock freq. = 1.23 × AMD clock freq.

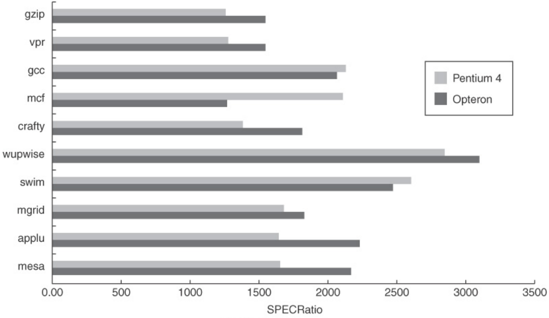


© 2007 Elsevier, Inc. All rights reserved.

Pentium 4 @ 3.2 GHz vs. AMD Opteron @ 2.6 GHz

Puttint it all together

- Can a processor with a lower clock frequency outperform a processor with a higher clock frequency?



© 2007 Elsevier, Inc. All rights reserved.

Conclusions

How do we compare processors?

- Lower CPI does not necessarily yield faster processors
- Processors with higher clock frequencies are not necessarily faster
- Instruction-level parallelism faces various limitations (walls):
  - Power wall – exponentially increasing complexity
  - Memory wall – non-overlapped memory latency
- What are we looking at next?
  - Software support for exploiting ILP
  - Designing more effective memory systems (caches, DRAM)
  - Looking in other sources of parallelism (threads, processes)