Instruction Level Parallelism

Software View of Computer Architecture COMP9244

Godfrey van der Linden

2006-04-06

Introduction

- Definition of Instruction Level Parallelism(ILP)
- Pipelining
  - Hazards & Solutions
- Dynamic Scheduling
  - Hazards & Solutions
  - Limitations of ILP
  - Power Consumption
- History
- Conclusion

Definition of ILP

- Early processors would use more than one cycle to execute an instruction. Instruction per cycle (IPC) < 1
- Pipelines overlap instruction execution to achieve IPC = 1
- ILP can be defined as - IPC > 1

Background - Pipelining

- RISC style instruction set architectures allowed hardware designers to vastly simplify the implementation of CPUs
- For example the MIPS instruction set architectures uses a maximum 5 cycle to implement any instruction
- RISC was designed for ease of pipelining

Pipelining (cont)

- A pipeline overlaps instructions to complete one instruction per cycle

	1	2	3	4	5	6	7	8	9
i	IF	ID	EXE	MEM	RWB				
i+1		IF	ID	EXE	MEM	RWB			
i+2			IF	ID	EXE	MEM	RWB		
i+3				IF	ID	EXE	MEM	RWB	
I+4					IF	ID	EXE	MEM	RWB

Pipelining - Hazards

- Pipelines are an easy extension to multi-cycle processor design, use the cross cycle latches to cross instruction phase
- There are some problems to be solved, aka pipeline hazards
  - Structural Hazard
  - Read after write(RAW) Hazard
  - Branch Hazard

Pipelining - Hazard Solutions

- Stall pipeline until hazard clears
- RAW Hazard - forwarding from one output phase direct to input phase
- Structural Hazard - duplicate units, eg. Split data & instruction caches
- Branch Hazard - predict a branch not taken and no-op & fetch if wrong

Instruction Level Parallelism (ILP)

- Higher performance means more instructions per second
- Implies higher clock rate or more instructions per clock cycle
- More instructions per cycle after a pipeline is IPC > 1, hence instructions need to complete in parallel

ILP (cont)

- Higher clock rates requires more, simpler & shallower pipeline stages
- However the simple pipeline hazard solutions are less efficacious
  - Branch mis-predicts require more stalls
  - RAW hazards need to be forwarded backwards in time
  - Structural units need to be split and reproduced for shallower phases

ILP Dependencies & Hazards

- True Data Dependencies
  - If inst j needs result of inst i, then j depends on i
- Name Data Dependencies
  - If j writes to reg a, and an earlier i reads a then order must be maintained (WAR)
  - If j writes to reg a, and an earlier i writes a then order must be maintained (WAW)
- Control Dependencies
  - Instructions streams are dependant on branch results

Dynamic Scheduling

- Technique to execute instructions as soon as dependencies are satisfied
- Scoreboarding tracks instructions on a functional unit scheduling execution when data dependencies are satisfied
- Tomasulo extends this with dynamic renaming of registers to clear name dependencies

Tomasulo Algorithm

- 3 Phase instruction execution
  - *Issue* - ID, if reservation station(RS) available schedule inst with current operands and dependencies
  - *Execute* - Operands available schedule instruction on functional unit
  - *Write Results* - On completion write results to register file and any RS that depends on it

Superscalar Processors

- Given a Tomasulo architecture can now add functional units to achieve even more parallelism
- Power4 has 2 Load/Store, 2 Fixed Point, 2 Floating Point, Branch and a CR unit

Branch Prediction

- 1 in 3 - 7 inst's is a branch. Control dependency stalls destroy IPC
- Modern unit - tournament between local and global branch predictors
- Stack of targets and returned addresses
- Branch prediction, today's typical unit achieves 95% accuracy
- OO virtual member functions present real problems to branch predictors without value prediction

Speculation

- Extend Tomasulo approach by adding a Reorder buffer (ROB)
  - Changes *issue* phase as ROB must be allocated
  - Adds *commit* phase, current head of the ROB FIFO is 'committed'
  - On a committing failed branch, flush ROB
- Solves precise exception problem too!

Ideal CPU ILP Limits

- What is maximum ILP achievable on an ideal machine

SPEC Benchmark	Average Inst. Issued
gcc	55
espresso	63
li	18
fpppp	75
doduc	119
tomcatv	150

Realisable CPU ILP Limits

SPEC/ Wdw	Inf.	256	128	64	32	16	8	4
gcc	10	10	10	9	8	6	4	3
espresso	15	15	13	10	8	6	4	2
li	12	12	11	11	9	6	4	3
fpppp	52	47	35	22	14	8	5	3
doduc	17	16	15	12	9	7	4	3
tomcatv	56	45	14	22	14	9	6	3

Power Consumption

- [Li03] found IPC/Power were correlated for OS routines on a superscalar
- Using a cycle accurate power simulator of MIPS R10000, 50% of power used by data-path & pipeline structure
- The cite 3 other papers that collaborating their findings

History of ILP

- ‘59 IBM 7030 “stretch” - Pipelining
- ‘64 CDC 6600 dynamic scheduling using scoreboarding
- ‘67 IBM 360/91 for dynamic scheduling Tomasulo
- ‘94/’95 1st gen superscalars: Pentium, AMD K6, MIPS 12000, PowerPC620
- End ‘90s 2nd gen: PIII, Athlon, Power4, Alpha 21264

Conclusions

- Modern superscalar speculative processors are extremely capable and complex
- Penalties for missed branches are large, OO language techniques are a problem, until IF phase can read registers
- Other penalties are also significant, static scheduling for a processor architecture of instruction stream is recommended

Conclusions (cont)

- Statically scheduling code for processors is very difficult, use compiler technology from chip manufacturer
- Software engineers in high level languages can not influence code scheduling, except with branch hints
- With profiling help an engineer can achieve some 5x performance improvement over generated code in statically scheduled assembler [Grey05]