Static Scheduling, VLIW, EPIC & Speculation

Today's topics:

HW support for better compiler scheduling

VLIW/EPIC idea & IPF example

Beating the IPC=1 Asymptote

- **Superscalar**
  - static/compiler scheduled
    - » common in embedded space MIPS & ARM
  - dynamic scheduled
    - » HW scheduling via scoreboard/Tomasulo approach
- **VLIW**
  - long instruction word contains set of independent ops

Int/Br	Int/Ld-St	FP+/-	FPmul/div

  - key – compiler schedule and hazard detection (εδ adv.)
    - » each slot goes to a particular type of XU
      - similar to reservation station role
  - problem in high performance practice
    - » need to be conservative w.r.t. run time activities
      - data dependent branch predicate
    - » fix – add some HW to make less conservative but probable choice

VLIW History

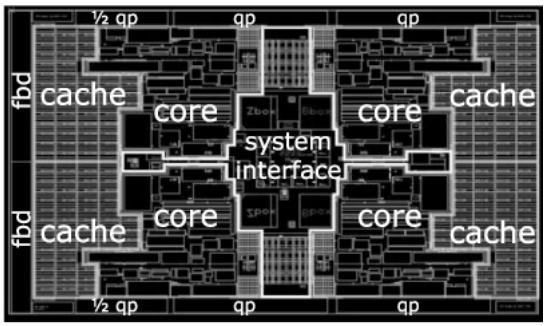
- As usual it's not new
  - late 60's early 70's – microcode
    - » same idea, different granularity
  - 80's (textbook inaccurate on this)
    - » Cydrome Cydra-5 (Rau/UIUC) & Multiflow (Fisher/Yale)
      - mini-super segment (Cray like performance on a budget)
      - killer micro ate them and the companies cratered
      - both Rau and Fisher go to HP to develop PA-WW
        - note both were compiler types (Fisher inspired by dataflow geeks)
  - 90's
    - » HP wants out of process business, Intel wants a server line
    - » HP & Intel jointly develop and produce Itanium
      - 2001 first release of "Merced" & IA-64
  - Now
    - » AMD shocks x86 land w/ 64-bit architecture at MPF 2000
    - » poor IA-64 integer performance forces Intel to follow suit
    - » IA-64 → IPF still happening
      - now all Intel but "Itanic" problems persist

"Itanic"

- Interesting quotes:
  - John Dvorak (journalist) article
    - » "How the Itanium killed the Computer Industry"
  - Ashlee Vance (tech columnist)
    - » underperformance + product delays
      - "turned the product into a joke in the semiconductor industry"
  - Donald Knuth
    - » "supposed to be terriffic – until it turned out that the wished-for compilers were basically impossible to write"
- However
  - illustrates some interesting architectural tactics
    - » approach highly valued in the embedded space
  - Tukwila (4 core IPF)
    - » "what rhymes with Godzilla and has enough cache to take out Tokyo?"
    - » 4 FB-Dimm channels
      - a move to dominate data-center now called "Cloud" apps

Tukwila

30MB L2$ total



34 GB/s memory b/w
96 GB/s skt-skt b/w

	FET count	Voltage	Power
Core logic	430M	0.9-1.15V	100W
Sys Int	157M	0.9-1.15V	30W
L3 cache	1,420M	1.10V	20W
IO logic	39M	1.10V	20W
Chip Total	**2.046B**		**170W**

2 threads/core target delivery "real soon now" original target was 2007

OUCH

QPI is Intel's response to AMD Hypertransport, 2 fbd's missing on RHS

VLIW Achilles' Heel

- **Code compatibility**
  - **backwards compatibility**
    - » always a bit of a boat anchor
  - **compiler schedules but what if the machine changes and you don't have the source code?**
  - **oops**
- **Solutions**
  - **Transmeta approach**
    - » dynamic object code translation
      - not wildly different than VM + dynamic issue
  - **IPF approach**
    - » don't be devout about VLIW
      - add some hardware support to allow some dynamic information

Itanium Example

- Registers
  - 32 64-bit + poison bit flag GPR's
  - 128 82-bit FPR's
    - » 2 extra exponent bits over IEEE 754 80-bit standard
  - 64 1-bit predication flags (single register)
  - 8 64-bit indirect branch registers
  - large set of special purpose regs
    - » I/O, system, memory map, OS interface
    - » rich set of performance counters
- Register stack
  - 128 architected registers
    - » 0-31 are the GPR's
    - » 32-127 are on the stack (cached or not)
      - special HW handles overflow and underflow
    - » special instructions manipulate stack frame save and restore

IPF Instructions and Slots

- Instruction types
  - A = int ALU
  - I = shifts, bit-tests, moves
  - M = memory access
  - F = floats
  - B = branches
  - L+X = extended immediates, stop, nops
- Instruction word slots
  - I = A or I types
  - M = A or M types
  - F = F types
  - B = B types
  - L+X = L+X types

IPF Groups and Bundles

- Instruction group
  - set of parallel instructions
  - arbitrary length w/ explicit stop bit
- Instruction bundle = 128 bits
  - a subset of a group that gets executed/cycle
  - contains pre-decode tag
    - » 5 bits indicates what the bundle order contains what the bundle contains
      - permutations (5,3) = 20 → 5 bits
    - » 3 41-bit instructions in the bundle
  - 2 bundles decoded and executed per cycle
    - » on Merced and McKinley
    - » key:
      - compiler generates the group and organizes code into bundles
      - HW decides decode and issue rate

IPF Predication and Speculation

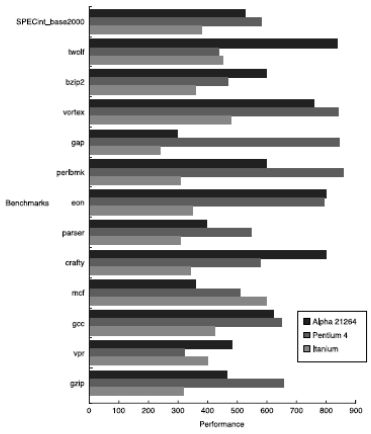
- Most instructions predicated on a predicate flag
  - 10 compare types
    - » result goes to 2 predication flags (dual rail encoding)
- Speculation
  - GPR's have a poison bit (indicating data validity)
    - » Intel calls them NAT's (Not a Thing)
  - FPR's indicate poison by NATVal
    - » mantissa=0, exponent outside legal range
      - hence the extra exponent bits
    - » interesting choice
  - advanced loads
    - » loads promoted over stores
      - return value to ALAT table (value, dest. reg, and mem. addr)
    - » if a previous store executing later matches mem addr
      - ALAT invalidated, and register poisoned
    - » interesting wrinkle on more common write buffer

IPF Pipe

- XU's
  - 2 I's, M's, F's, 3-B's, 1 L+X
- Issue 2 bundles = 6 instructions max
- Pipe – 10 macro stages
  - IPG – prefetch 2 bundles
  - Fetch – decode
  - Rotate – rotate bundle to align the stops
  - EXP – hand instructions to the XU's – issue
  - REN – rename registers
  - WLD – bypass and access reg. file
  - REG – checks register scoreboard dependencies (dynamic stall if not cleared)
  - EXE – execute
  - DET – detect exceptions and post NAT's
  - WRB – write back

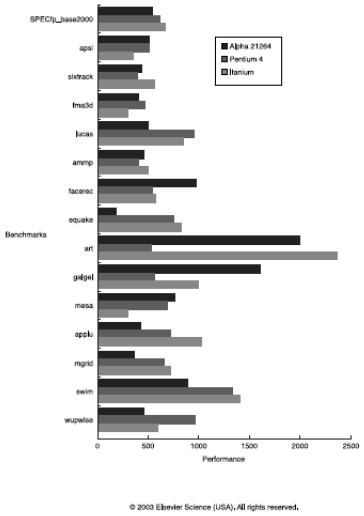
Merced SpecInt Performance

based on
800 MHz
Itanium



OOPS!!

SpecFP is Better



Newer versions close integer gap but x86 is still better

New Tukwila focus on memory and socket to socket interconnect may prove to win

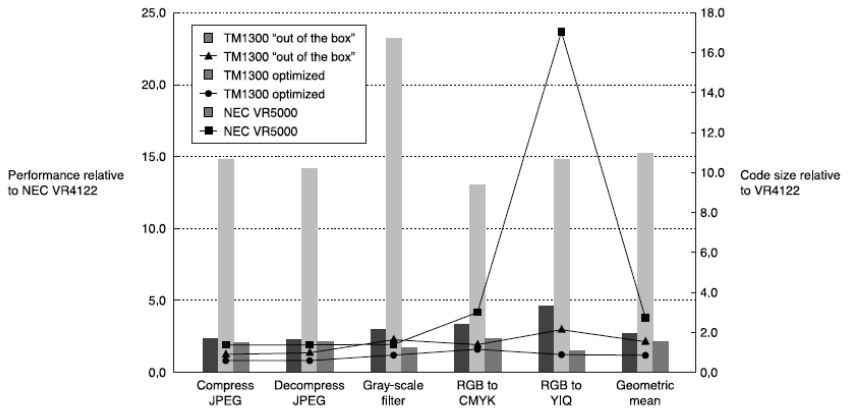
BUT 8 core Nehalem waits in the wings with QPI as well

Time will tell

Philips Trimedia TM32

- Pure VLIW for embedded space
  - no HW hazard detection
    - » compiler does all
    - » saves runtime energy and delay
  - no virtual memory
    - » loads/stores don't generate exception
      - no TLB and alignment issues
  - main problem
    - » code bloat
      - instruction memory is limited in embedded devices
      - contribution to leakage current is a potential problem
    - » pure SW schedule
      - large number of explicit NOPs

TM32: Performance on EEMBC



Transmeta Crusoe

- **Dynamic code morphing**
  - **Boris Babayan's idea (St. Petersburg IPM)**
  - **x86 to VLIW in front end**
    - » **table based so adds post manufacture flexibility**
      - **and some fault tolerance**
    - » **5 slots: risc style IU, compute (IU, FU, multi-media U), memory, branch, immediate (32 bits used by another instruction)**
    - » **4 ops/128-bit-slot**
      - **2 options**
        - **memory, compute, ALU, immediate**
        - **memory, compute, ALU, branch**
- **Speculation support**
  - **shadowed register files**
  - **program controlled store buvver**
    - » **SW controlled commit with auto-rollback**
  - **speculative loads – (ALAT like)**
  - **conditional move instruction**
    - » **better than full predication IMHO**

Crusoe Performance

- Weak (in a word)
  - early netbook like devices
    - » 10 minutes to boot up – UGHly
- But in terms of power per workload it looks good
  - MP3 playback
    - » 500 MHz, 1.6V Pentium M – 0.672 watts
    - » 400 MHz, 1.5V TM3200 = 0.214 watts (32% of M)
  - DVD playback
    - » M = 1.13 watts
    - » TM = .479 watts (42% of M)
  - difference
    - » energy/workload unit is better metric
      - power for continuous media processing isn't bad though
    - » TM3200 barely kept to real-time schedule
    - » Pentium III M – had much more headroom

Compiler Support for ILP

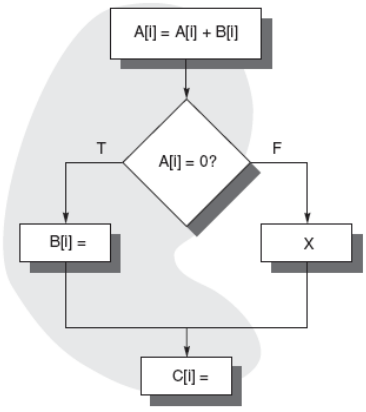
- Easier if:
  - non cyclic dependencies
    - » e.g. loop carried
  - recurrent dependencies
    - » dependency n loops away?
      - unroll n-1
  - affine array index calculations
    - » address = ai + b
      - i is loop index, a & b are constants
      - compiler can know when conflict exists
- Harder if:
  - indirect references
    - » via pointer
    - » via array of pointers – common in sparse matrix computations
  - false dependency
    - » for some data values a dependency may exist but it is rare
      - compiler has to be conservative

Compiler Techniques

- Already seen (focus on loops)
  - branch prediction, unrolling, SW pipelining
    - » note for static schedules branch prediction is a result of profiling
- Add trace scheduling (focus on conditionals within a loop)
  - 2 separate phases
    - » trace selection
      - predict multiple branches to give long sequence of instructions
      - each possible sequence is a separate trace
      - selection depends on how conditions actually resolve
    - » trace compaction
      - global instruction scheduling over entire trace
        - tricky bit: moving instructions across predicted branch
          - potential change to the controlled/uncontrolled requirement
      - speculation based on prediction
        - exceptions become an issue
        - if speculate wrong then clean-up is required
          - cost of clean-up becomes a consideration
          - as is probability of being correct

Start with Basic Block

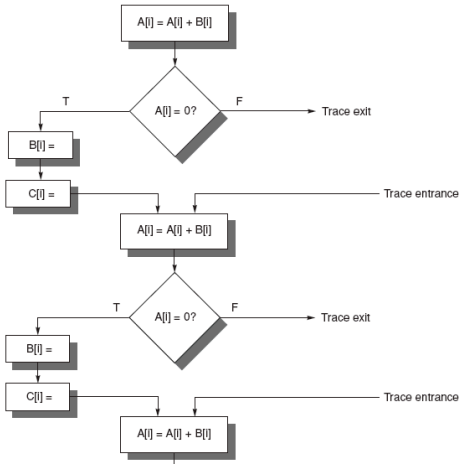
- if then else is in the inner loop



Profile indicates shaded path is the most common

Unroll 2+ → Trace

- **Problem is if the unpredicted path**
  - **trace now has multiple entries and exits**



No problem if predicted path happens most of the time

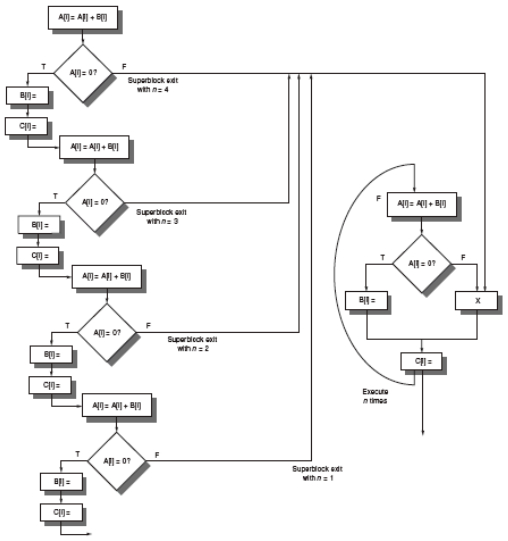
mutliple exit and entry is hard to schedule

compiler cost function is complex so it's hard to know what the best option really is

Superblocks

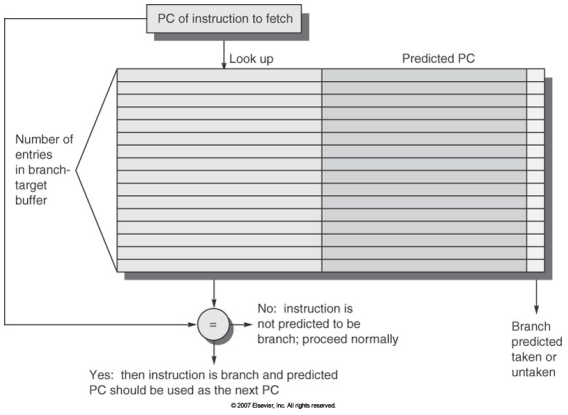
- **Trace-like idea**
  - **but single entry multiple exit**
    - » **code motion now only moves across exit**
    - » **on exit any moved and "should have been" controlled code must be compensated for**
  - **tail duplication**
    - » **handles any remaining body loops**
    - » **and compensates for the "should have been" code**

Example (no compensation)

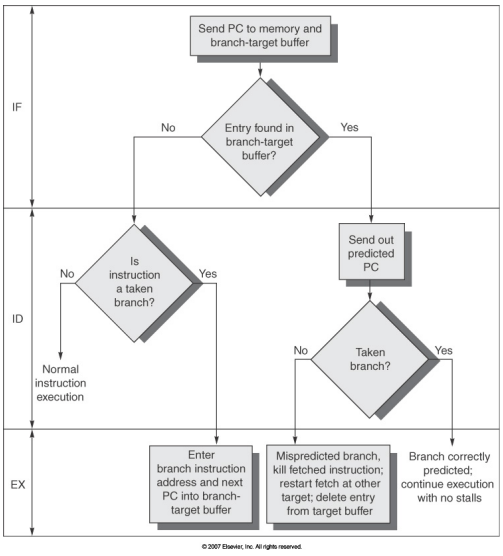


Improving on Branch Prediction

- **Branch Target Buffer (opus 1)**
  - **note books description is a bit strange (Fig 2.22)**
    - **why?**



Better Version (1 branch delay slot)



Branch Folding

- **BTB holds target instruction rather than PC**
  - **for n-issue this means n instructions**
    - » **can already hold decoded version rather than fetched version**
  - **creates 0 cycle branch delay**
  - **problem?**

Branch Folding

- BTB holds target instruction rather than PC
  - for n-issue this means n instructions
    - » can already hold decoded version rather than fetched version
  - creates 0 cycle branch delay
  - problem – you bet
    - » BTB is a cache so how does it get populated
      - likely want only active branches
        - e.g. in a loop
      - unlikely to significantly over provision fetch
    - » 2 choices
      - run ahead – fetch & cache taken path when you can
        - if instructions not cached then take the hit w// pipeline bubble
          - since compiler didn't try to fill the delay slot
      - or cache taken instructions when they appear
        - more likely scenario
        - this will always happen for unconditional branches
          - uncondix branches are stupid but an artifact of linear code stream
- Similar benefit for trace cache

Concluding Remarks

- At this point we've explored 2 multiple issue domains
  - superscalar and VLIW
    - » there is some overlap
      - e.g. what's the difference between n-issue static superscalar and pure n-wide VLIW?
        - A: almost nothing
    - » take home
      - If minimum energy is your key concern
        - let the compiler do what it can
        - the HW just follows the orders
      - If performance counts
        - then HW mechanisms will be required
          - depending on how far you go
            - It's possible to use a lot more energy for little performance gain
      - what you should care about
        - $\varepsilon\delta$ product
        - it's one good way to decide on design quality
          - add in frequency and you get power