Instruction Level Distributed Processors

J. E. Smith

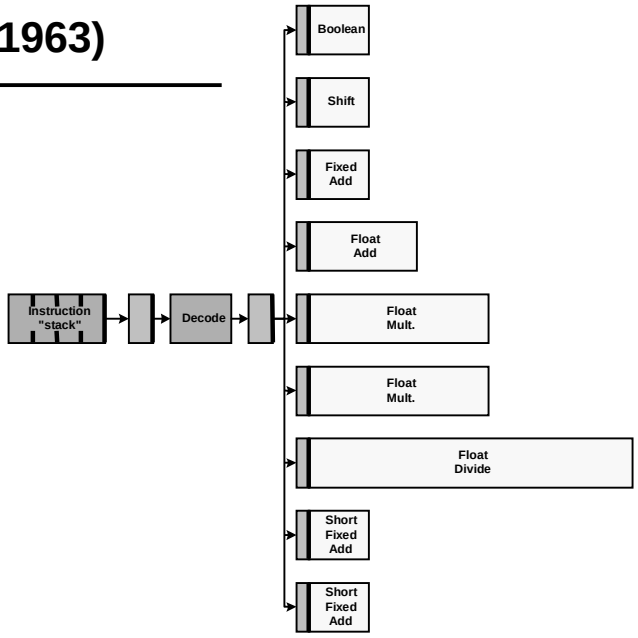
ECE Dept.

Univ. of Wisconsin-Madison

Future Design Challenges

- **Wire delays**
- **Power**
- **Design Complexity**
- **These are not new challenges**
  - **We can learn from the past… ?**
  - **WWSCD – what would Seymour Cray Do?**

CDC 6600 (1963)

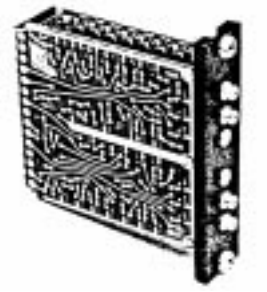


CDC 6600

- **Wiring**
  - "***Distances … can range up to about five feet, with transmission time of about 1.3 nanoseconds per foot. Since the circuit speed is in the range from three to five nanoseconds, this distance must must obviously affect the design***" **– J. E. Thornton**

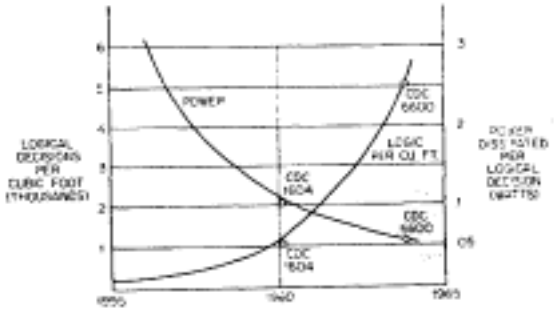
Cordwood packaging

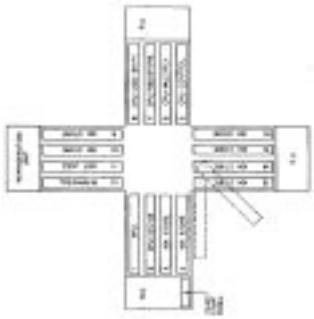
2 PC cards, face-to-face, with wires and resistors connecting in 3rd dimension



CDC 6600

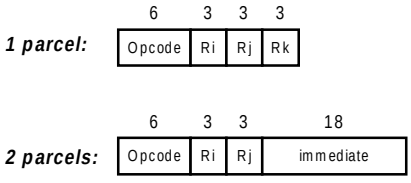
- **Power**
  - "***A particularly difficult set of problems centers around the power distribution and cooling methods… it has become necessary to examine alternatives to the traditional air cooling***"





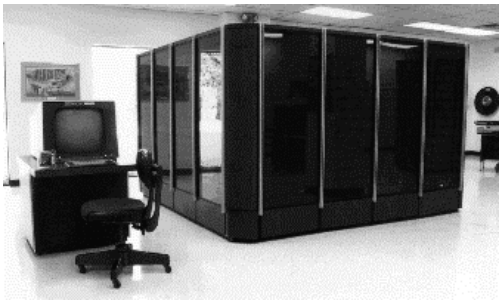
CDC 6600 Architecture Features

- **Instruction set and implementation co-designed**
  - **Its all *hardware***
- **Small register files**
- **Sequential processing elements**
- **Variable sized instructions**



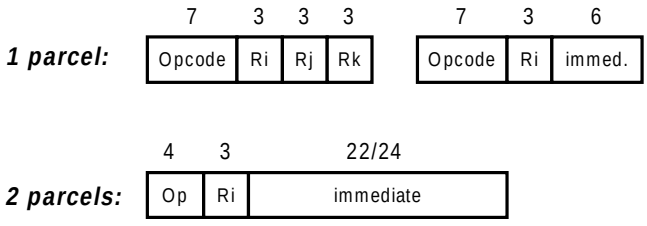
CDC 7600 (1969)

- ISA similar to 6600
- Pipelined
- Simple, in-order instruction issue
  (based on 6600 experience)

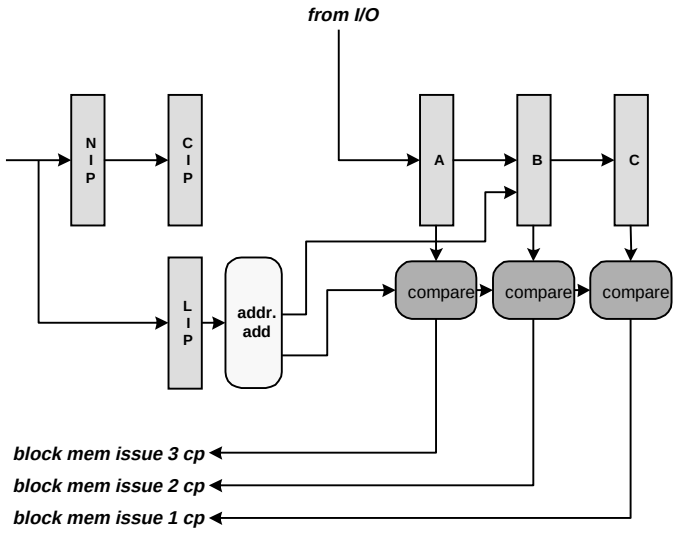


Cray-1 (1976)

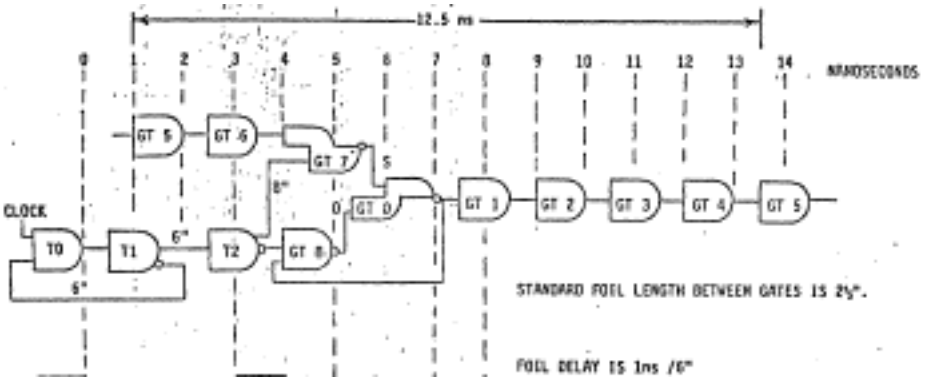
- Hierarchical Register Files (8,64)
- In-Order issue
- Put all stalls in issue stage (even memory conflicts)
- Minimize speculation and arbitration



Cray-1 Memory Conflict Checking

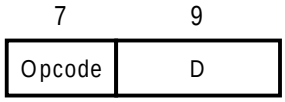


Cray-1 Critical Path



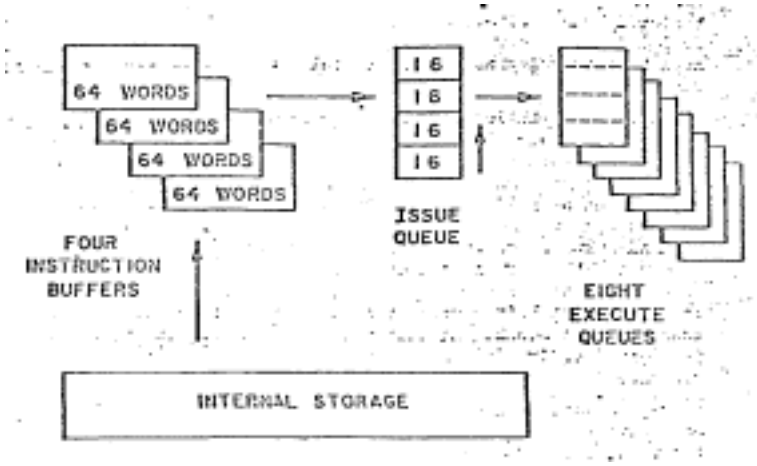
Cray-1.5 (circa 1979?)

- **Never Finished**
  - **Vestiges in 1S I/O processor**
- **Hierarchical register files (1,512)**
  - **Accumulator instruction set**
  - **One instruction length (16 bits)**
  - **Branch flag**
  - **Immediate values in memory**



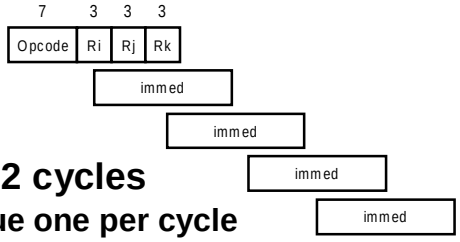
Cray-1.5 Microarchitecture

- **Structured around dependence chains**
  - **rename accumulators**



Cray-2 (1985)

- Cray-2 used 4 gate levels between clocks
  - And 2-phase clock
  - ⇒ It was all latches!
  - integer add – 8 cp
  - taken branch –17 cp



- Max issue rate 1 per 2 cycles
  - Cray-3 managed to issue one per cycle
- Non-arbitrated memory interconnection network

Processor performance gains

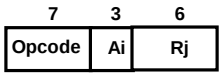
- About 4% per year due to microarchitecture innovation.
  - *Innovation didn't begin with microprocessors!*
    - deep pipelining began in mid 60s
  - Major performance gains have been from technology
- Modest Proposal:
  - Design a simple fast general purpose core, and call it a *solved problem*

Instruction Level Distributed Processing

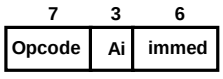
- Keep everything small and simple
  - Sequential processing elements
  - Hierarchical register files
- Focus on dependences and communication
  - Accumulator for chaining instructions
  - Account for interconnect as a part of the microarchitecture
- Develop ISA and microarchitecture together
  - Use VM-based dynamic binary translation
- Distributed Processing Elements
  - Suitable for GALS clocking
  - Enable large-scale clock/power gating

Instruction Set

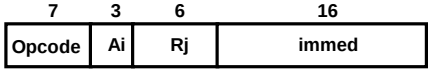
- 8 accumulators
- 64 (or 32) general purpose registers (GPRs)
- Formats:



register format



short immediate format



long immediate format

Instruction Set

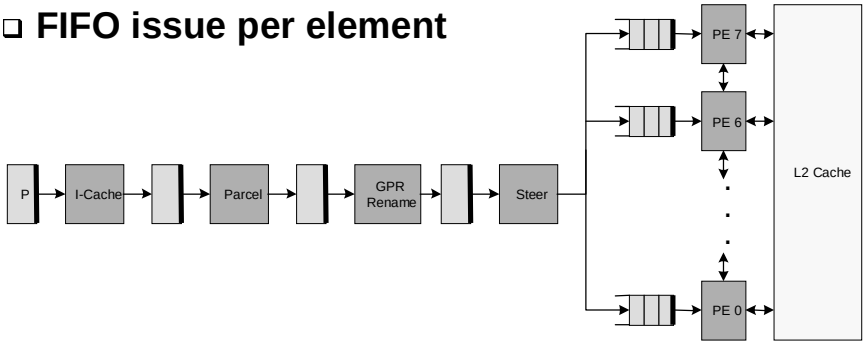
- **Load/Stores**
  - `Ai <- mem(Ai)`
  - `Ai <- mem(Rj)`
  - `mem(Ai) <- Rj`
  - `mem(Rj) <- Ai`
- **Register operations**
  - `Ai <- Ai op Rj`
  - `Ai <- Ai op immed`
  - `Ai <- Rj op immed`
  - `Rj <- Ai`
- **Branches/Jumps**
  - `P <- P + immed; Ai pred Rj`
  - `P <- P + immed; Ai pred 0`
  - `P <- Ai`
  - `P <- Rj`
  - `P <- Ai; Rj <- P++`

Example (from *gzip*)

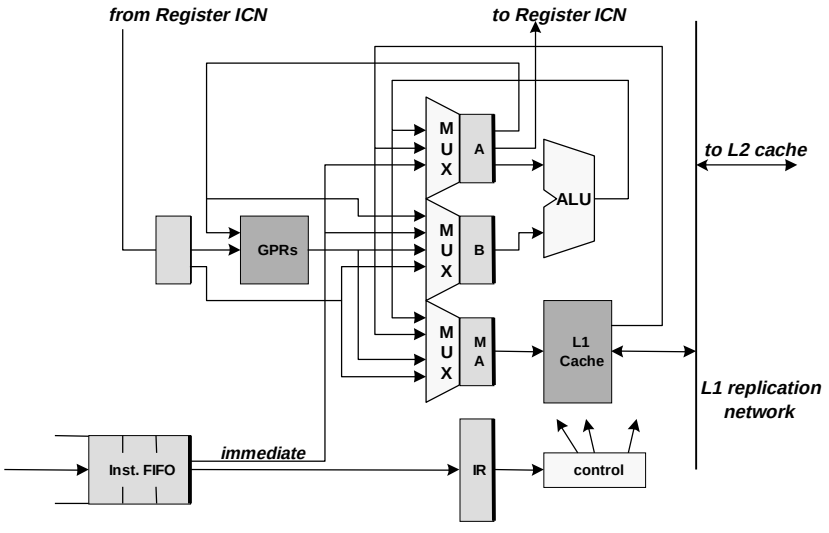
Alpha assembly code	Equivalent RTL notation	ILDP code
`L1: ldbu t2,0(a0)`	`L1:R2 <- mem(R0)`	`L1: A0 <- mem(R0)`
`subl a1,1,a1`	`R1 <- R1 – 1`	`A1 <- R1 – 1`
		`R1 <- A1`
`lda a0,1(a0)`	`R0 <- R0 + 1`	`A2 <- R0 + 1`
		`R0 <- A2`
`xor t0,t2,t2`	`R2 <- R2 xor R8`	`A0 <- A0 xor R8`
`srl t0,8,t0`	`R8 <- R8 << 8`	`A3 <- R8 << 8`
		`R8 <- A3`
`and t2,0xff,t2`	`R2 <- R2 and 0xff`	`A0 <- A0 and 0xff`
`s8addq t2,v0,t2`	`R2 <- 8*R2 + R9`	`A0 <- 8*A0 + R9`
`ldq t2,0(t2)`	`R2 <- mem(R2)`	`A0 <- mem(A0)`
`xor t2,t0,t0`	`R8 <- R2 xor R8`	`A0 <- A0 xor R8`
		`R8 <- A0`
`bne a1, L1`	`P <- L1, if (R1 != 0)`	`P <- L1, if (A1 != 0)`

Microarchitecture

- Unified instruction fetch/decode
- Distributed processing element(s)
- *Strands* are steered to PEs
- FIFO issue per element



Processing Element



Instruction execution

FIFO 0	Issue cycle
A0 <- mem(R0)	0
A0 <- A0 xor R8	1
A0 <- A0 and 0xff	2
A0 <- 8*A0 + R9	3
A0 <- mem(A0)	4
A0 <- A0 xor R8	5
R8 <- A0 6	

FIFO 1	
A1 <- R1 – 1	0
R1 <- A1	1
P <- L1, if (A1 != 0)	2

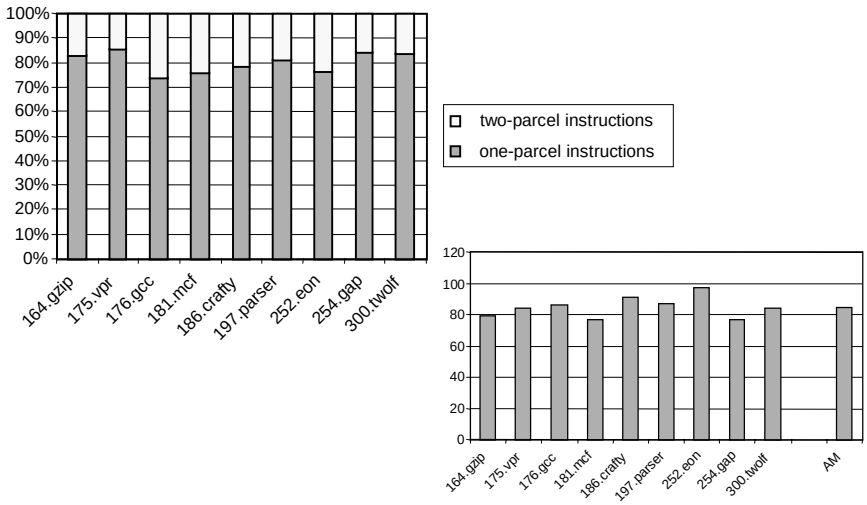
FIFO 2	Issue cycle
A2 <- R0 + 1	0
R0 <- A2	1

FIFO 3	
A3 <- R8 << 8	1
R8 <- A3	2

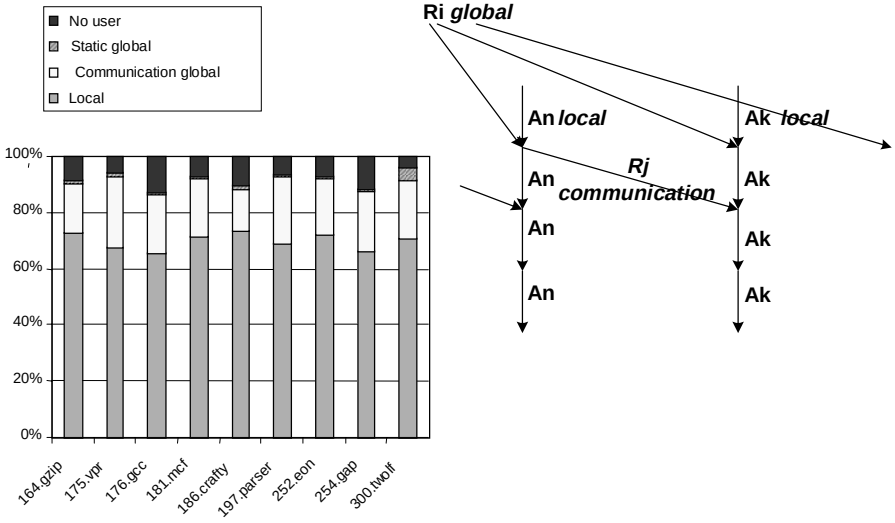
Evaluation

Benchmark	Alpha Insts	% zero or one input register	% of loads w/o immediate	% of stores w/o immediate
164.gzip	3.50 billion	55.09	43.6	50.6
175.vpr	1.54 billion	51.16	34.9	29.1
176.gcc	1.89 billion	62.38	34.8	15.8
181.mcf	260 million	57.74	30.4	11.9
186.crafty	4.18 billion	54.34	27.0	13.4
197.parser	4.07 billion	57.68	44.8	22.2
252.eon	95 million	55.83	15.7	15.4
254.gap	1.20 billion	61.60	44.9	27.1
300.twolf	253 million	50.48	41.5	31.2

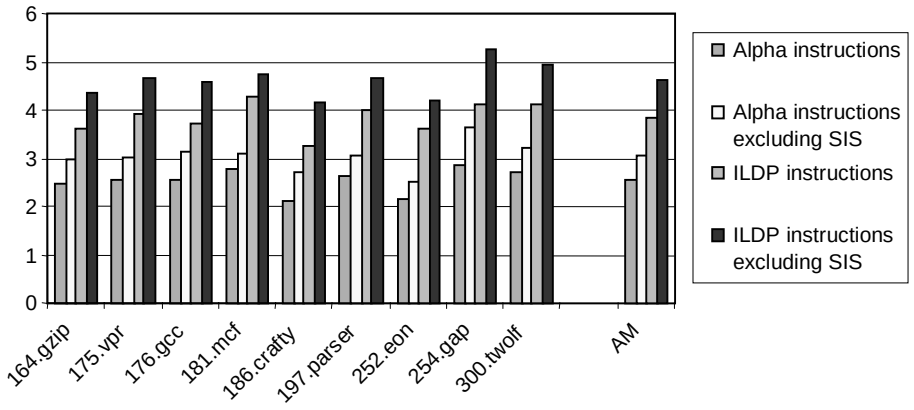
Instruction Sizes



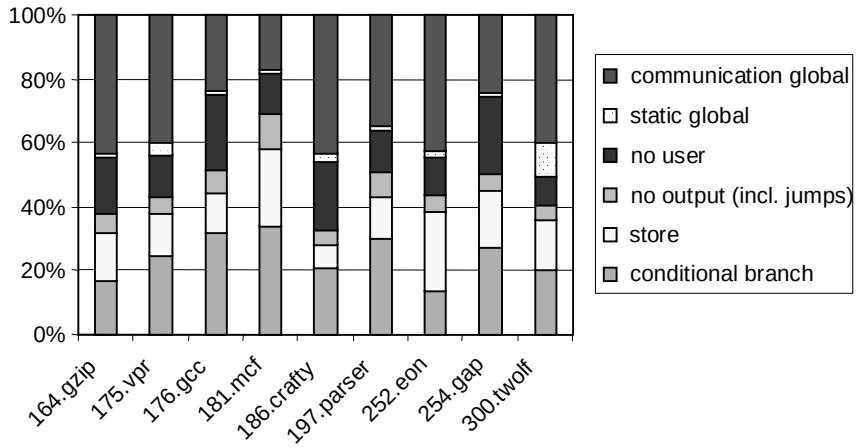
Register Usage



Strand Lengths

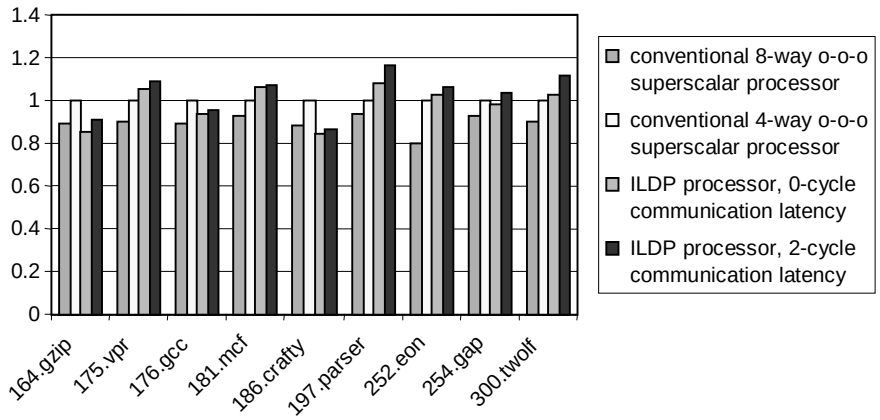


Strand Ends



Performance Results

- **Relative Performance**



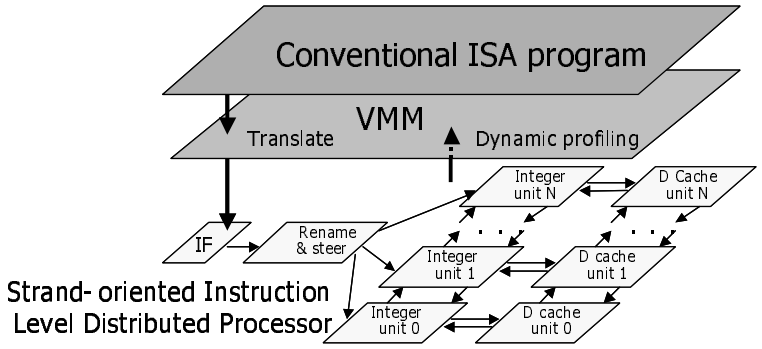
Complexity

	I-Fetch	Rename	Issue	Reg. Read	Units	Bypass
4-wide superscalar	**128 bits**	**12 R+W ports**	**4-way ooo 32-window**	**12 R+W ports**	**pipelined**	**Mx8**
4-wide ILDP	**64 bits**	**4 R+W ports**	**1-way io x8**	**3 R+W ports**	**sequential**	**Nx2**

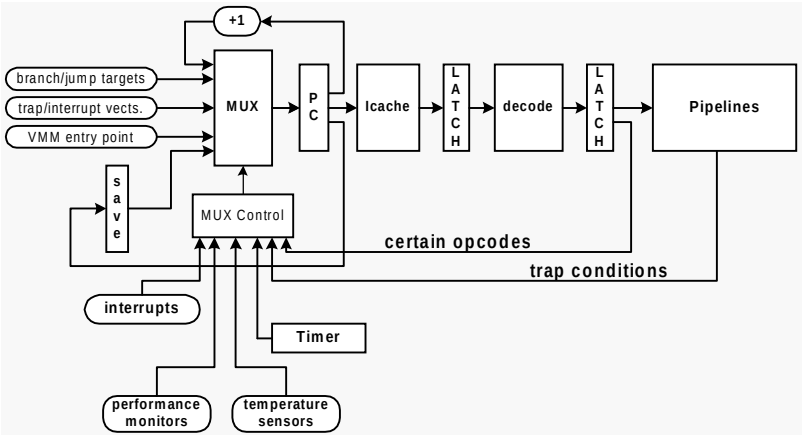
M>N

Co-Designed VMs

- **Use dynamic binary translation**
  - **Similar to Crusoe, Daisy**
  - **But, *Co-Designed VM != VLIW***

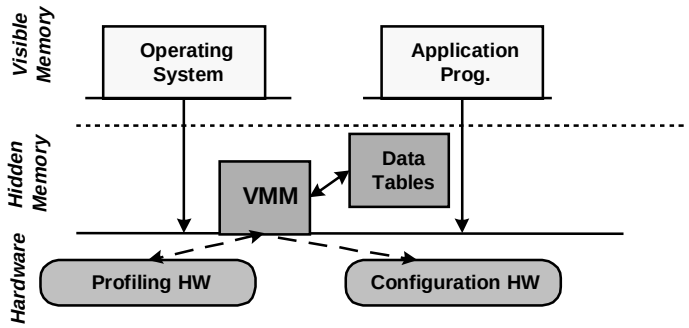


Microarchitecture

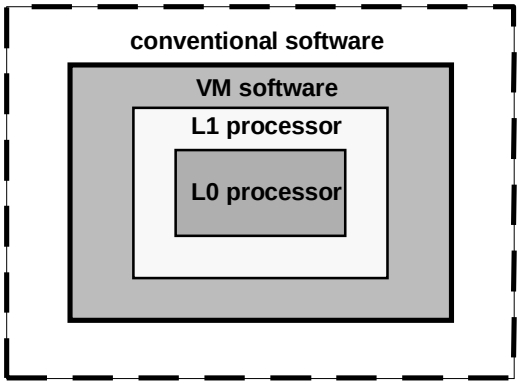


VMM as a "Micro Operating System"

- Manage *processor* with OS-like VMM software
  - Manage processor resources in an integrated way
  - Identify program phase changes
  - Save/restore implementation contexts
  - And, Microsoft doesn't need to know…



Another View



- "Solve" L0 processor problem
- Flexibility/innovation at outer levels

Additional Comments and Conclusions

- Red Giants => White Dwarfs
- ILDP designed for wire delays, simple logic
- On-chip multithreading – server applications
  - Replicate ILDPs
  - Even less need for single, *heavyweight* processor
- High ILP applications, e.g. multimedia, games
  - Do it the right way – use *vectors*
- GALS
- Power management at microarchitecture level
- Performance issues: branch prediction, cache misses
- Future work
  - Translator (x86?, IA64?, PowerPC?)
  - Gate level design of key parts

Acknowledgements

- Graduate Students
  - Jason Cantin
  - Ashutosh Dhodapkar
  - Timothy Heil
  - Shiliang Hu
  - Tejas Kharkanis
  - Ho-Seop Kim
  - Marty Licht
  - Subbu Sastry
  - Brandon Schwartz
- Funding
  - NSF, SRC, IBM, Intel