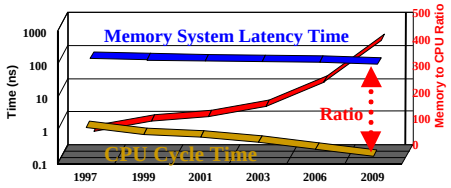
The State of *State*

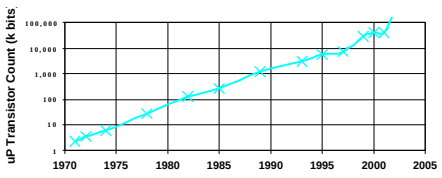
Peter M. Kogge
McCourtney Prof. of CS & Engr,
Assoc. Dean for Research, University of Notre Dame
IBM Fellow (ret)

Motivation

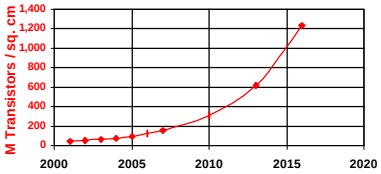
- **The "Memory Wall:"**



- **Relentless Moore's Law**



- **With more on the way**



Law of Unintended Consequences

- Rapid growth of "State"
  - Of many flavors and layers
- Forcing us to bigger & bigger chips
  - That is further & further from memory
- Triggering work on more sophisticated solutions
  - That expand state even further
- Condemning *key* state components to be buried even further from memory

This Talk

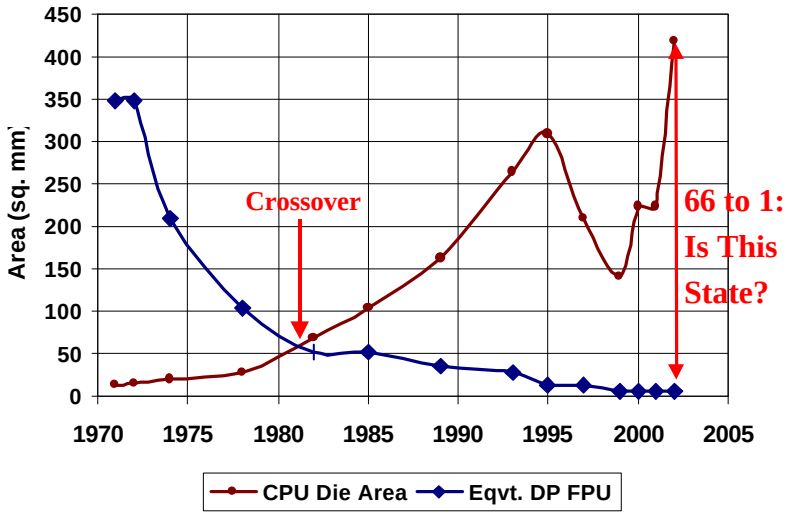
- Review all the "states" of state
- Correlate with architectural "advances"
- Analyze "costs" we are paying
- Discuss current "hot techniques"
- Explore alternatives

Disclaimer

All statistics used in this presentation are solely my own estimates, derived from open sources, and may not correspond to reality.

They are given here
for discussion purposes only.

How Are We Using Our Silicon? Compare CPU to a DP FPU



State?

Dictionary (Funk & Wagnalls):

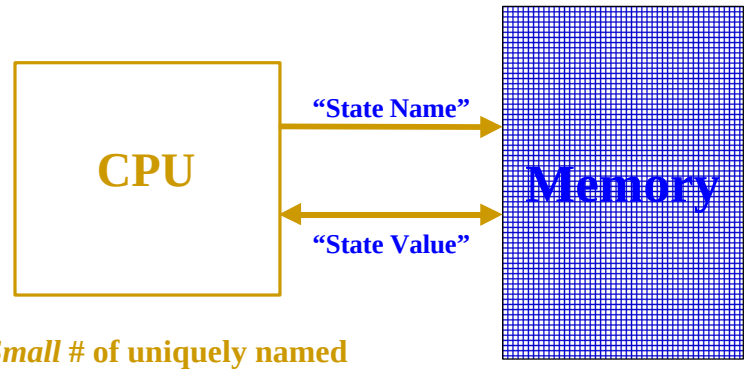
- **Mode of existence as determined by circumstances, condition, situation**
- **To set forth explicitly in speech or writing**
- **To fix; determine; settle**
- **Frame of mind; mood**

A Working Definition

State:

- **Those pieces of *named* information**
- **Whose particular *values***
- **May affect outcome of a program**
- **When executed on a computer**
- **That has access to the state value**

Von Neumann State



Small # of uniquely named registers of state that change *very frequently*

Large # of individually addressed bytes of state that change *relatively infrequently*

State of Interest Here!

States of State?

- *Purpose*: hold values, copies, or "meta-values"
- *Size*: amount of information in bits
- *Structure*: complexity of "sub-values"
- *Persistence* (lifetime): Infinite to sub cycle
- *Volatility*: Never change to constantly
- *Visibility*: which type(s) of software access it

Types of State

- *User*: state used for application execution
- *Supervisor*: state used to manage user state
- *Machine*: state that configures the system
- *Transient*: state used during instruction execution
- *Access-Enhancing*: state used to simplify translation of other state names
- *Latency-Enhancing*: state used to reduce latency to other state values

State Changing ISA Decisions

Assume on-CPU chip supported ISA features:

- Basic data width: 4, 8, 16, 32, 64
- Number of visible registers
- Support for virtual addressing
- Floating point
- Vectors/SIMD operations
- Multi-Threading

State Changing Microarchitectural Decisions

- Pipelining
- TLBs
- Hardware floating point
- Caches
- Branch Prediction
- Superscalar
- Out of order superscalar
- Debugging, performance counting

Eg. Superscalar

- More Transient bits in pipeline latches
- More instructions in flight
  - Larger Issue Pools
  - Bigger Physical Register Files, ROB, Rename Table
- More state for branch prediction
- More pressure on memory system
  - Multi-ported caches
  - Deeper cache hierarchy

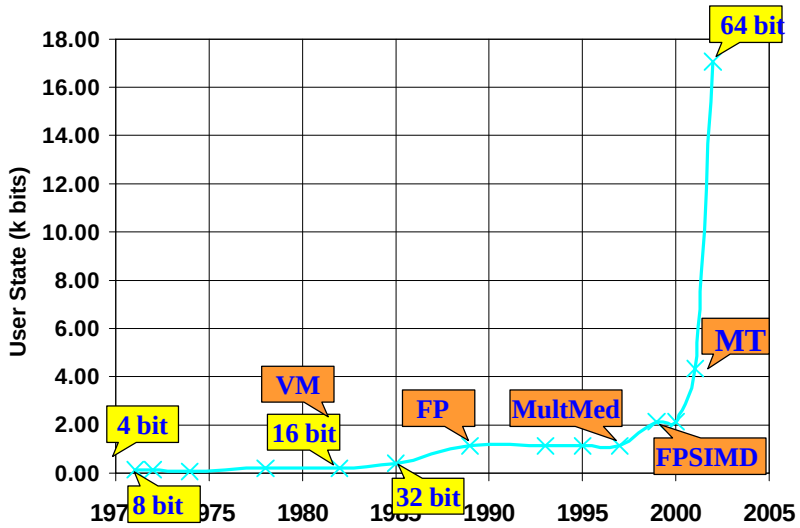
Eg: N-way Multi-Threading

- N x User state (N sets of ISA registers)
- Some Supervisor state
  - Depends on degree of thread separation
- Increase in Transient state
  - N x I TLB
  - N x prefetch buffers
  - Partial N x Branch History Buffer
  - N x Register Alias Table
  - Additional thread identifying tags per pipe stage, issue pool, ROB entry

User State

- *Purpose*: State used for application execution
- *Example*: Architectural Registers, PC, Flags
- *Size*: Depends on ISA
- *Structure*: Depends on ISA
- *Persistence*: Long
- *Volatility*: Changes on per instruction basis
- *Visibility*: Completely

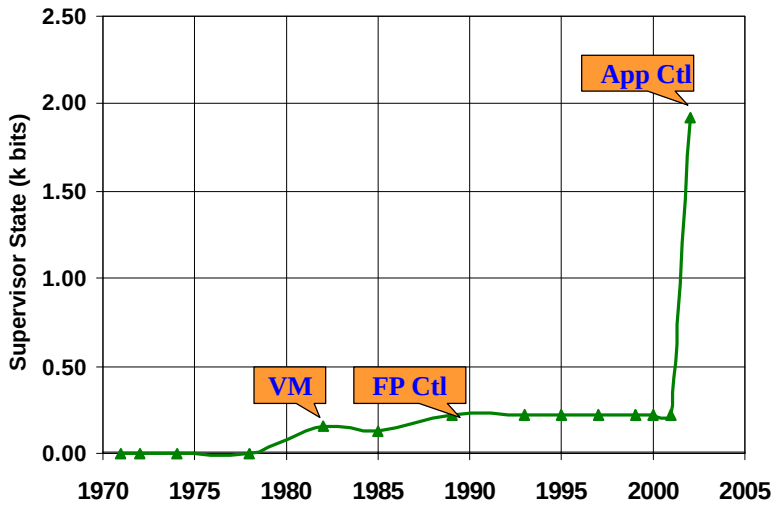
User State vs Time



Supervisor State

- ***Purpose*: State used to manage user state**
- ***Example*: Page Maps, Task Controls**
- ***Size*: Relatively small & independent of ISA**
- ***Structure*: Primarily pointer registers**
- ***Persistence*: Lifetime of a process**
- ***Volatility*: Small**
- ***Visibility*: Primarily OS**

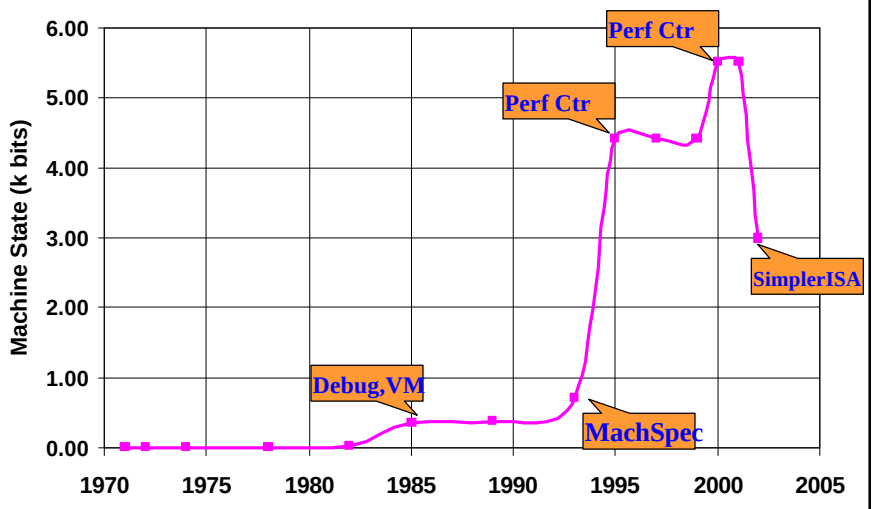
Supervisor State vs Time



Machine State

- ***Purpose:*** **Configure System, Support Debug & Test**
- ***Example*****: Boot-time setup, Breakpoint set, Performance counters**
- ***Size*****: Relatively independent of ISA**
- ***Structure*****: Somewhat complex**
- ***Persistence*****: As long as powered up**
- ***Volatility*****: relatively low**
- ***Visibility*****: Primarily boot & OS**

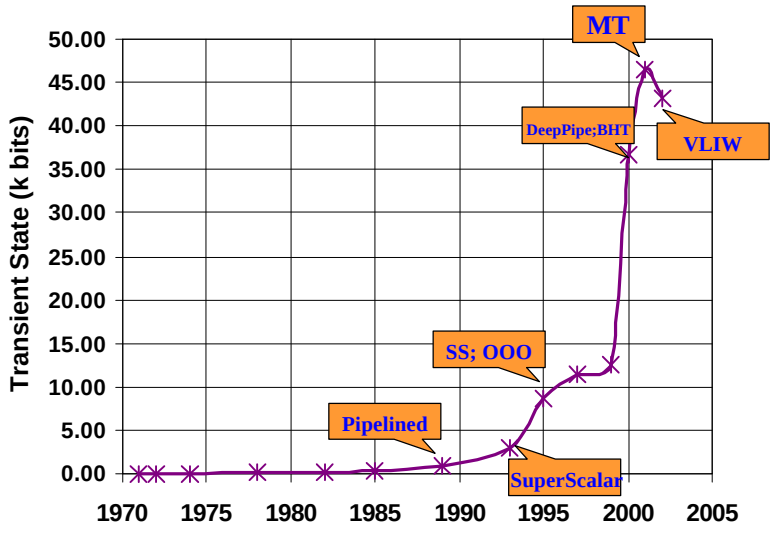
Machine State vs Time



Transient State

- ***Purpose:*** **Improve performance**
- ***Example*****: Pipe latches, BHT, ROB, ...**
- ***Size*****: Dependent on pipe depth, issue width & order**
- ***Structure*****: Very complex**
- ***Persistence*****: Cycle to thread timeframe**
- ***Volatility*****: very high**
- ***Visibility*****: invisible to software**

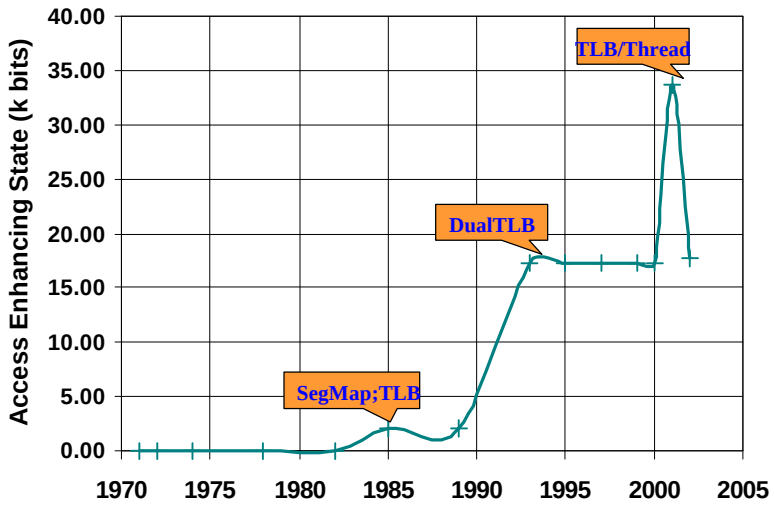
Transient State vs Time



Access-Enhancing

- ***Purpose*: State used to simplify translation of other state names**
- ***Example*: Segment mapping regs, TLB**
- ***Size*: Medium large**
- ***Structure*: Fairly regular**
- ***Persistence*: Cycle to process timeframe**
- ***Volatility*: moderate**
- ***Visibility*: Somewhat visible to OS**

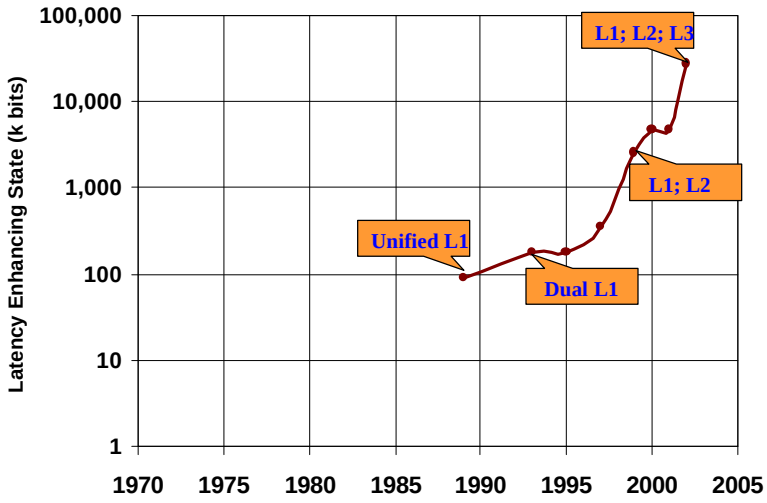
Access Enhancing State vs Time



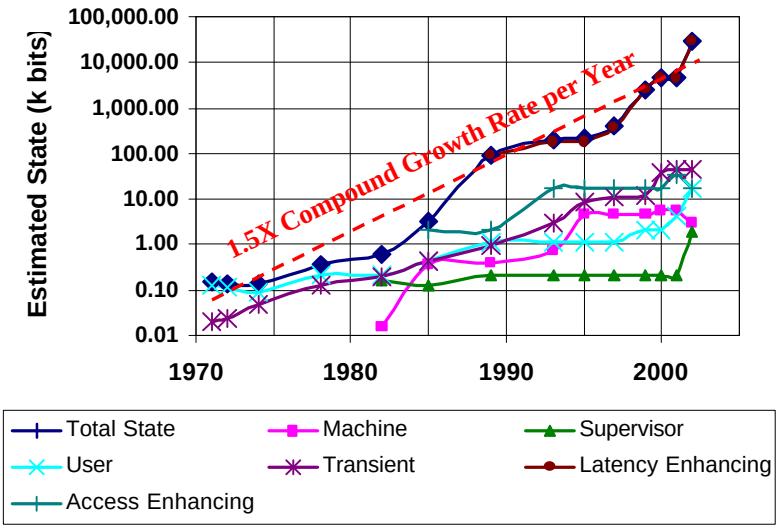
Latency-Enhancing

- ***Purpose*: State used to reduce latency to other state values**
- ***Example*: Caches, Vector Registers**
- ***Size*: Very large**
- ***Structure*: Very regular**
- ***Persistence*: process**
- ***Volatility*: moderate**
- ***Visibility*: through effects on performance**

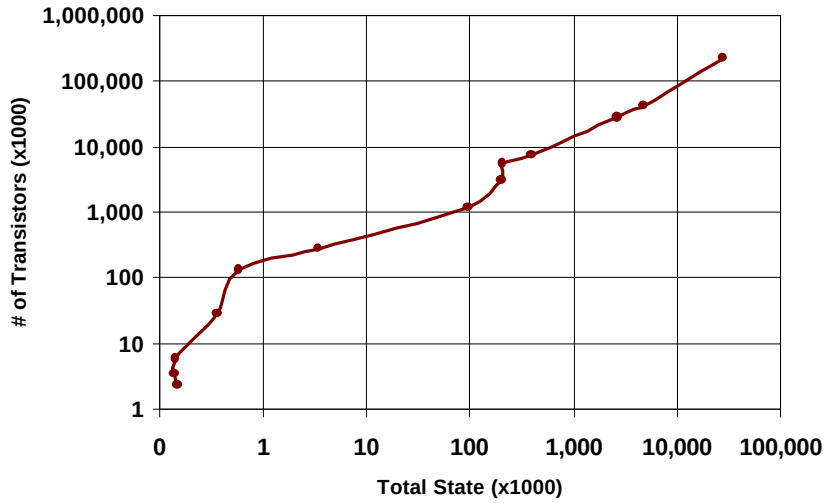
Latency Enhancing State vs Time



Total State vs Time



So We Expect State & Transistor Count to be Related



Power: The Other Byproduct of State

- Most State in SRAM Register File Format
- Often multi-ported
- Area grows as N^2 (N=# ports) x # bits
  - N often a function of issue width W (I.e.N=3W)
- Energy per Instruction grows as W^A

Structure	A
Register Rename Table	1.1
Instruction Issue Window	1.9
Memory Disambiguation	1.5
Physical Register File	1.8
Data Bypass	1.6
Functional Units	0.1
All Caches	0.7

Zyuban&Kogge,01

Effect on Power/Performance

- If Energy/Instr ~ W^A, A>1
- And IPC ~ $W^{1/2}$ (Amdahl's Law)
- Then Power ~ Energy/Instr x IPC x Clock
- And Performance ~ IPC x Clock
- Thus: Watts/Mip ~ W^A
  - *EXPLOSIVE GROWTH AS W INCREASES*

A Plea to Architects

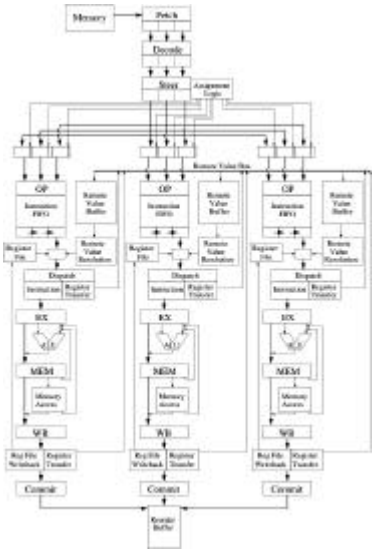
Relentlessly Attack **State Bloat**

by

Reconsidering **Underlying Execution Model**

Starting with Light Weight States

A Microarchitectural Trick: Multi-Clusters



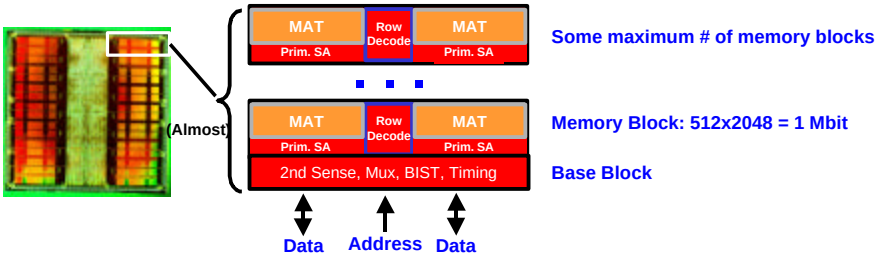
Replace large W-sized structures by m copies of < W/m - sized

- Fewer total transient state bits
- Simpler caches
- Power/MIP ~ $m(W/m)^A$
  ~ W^A/m^{A-1}
- Same performance @ 50% power
- or 20% more performance @ same power

A More Interesting Combination

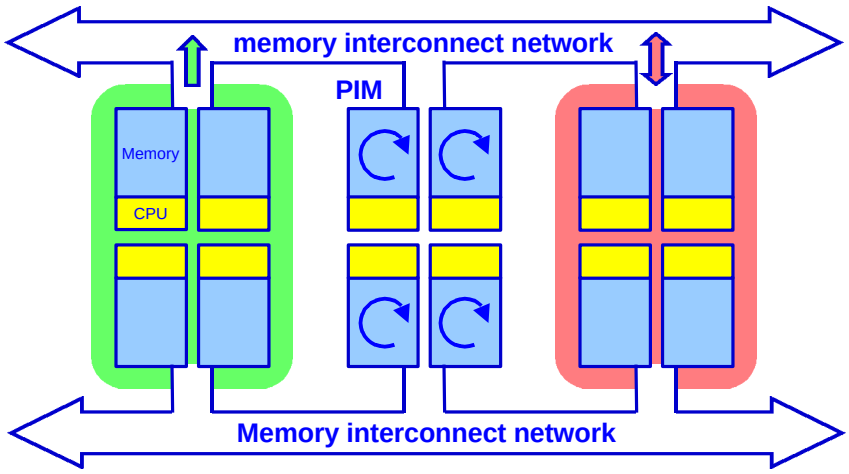
- Processing-In-Memory to reduce latency
- Light Weight Thread-Based ISA
- Memory Focused Execution Model

Embedded DRAM Macro



- Data called "wide word": often up to 256 bits/access
- Lo V_{th}: up to 16 memory blocks for 2MB macro
  - Access time: 5 ns; Page mode: 1.25ns
- Hi V_{th}: up to 64 memory blocks for 8MB macro
  - Access time ~ 40% faster

PIM As A "Memory Chip that's Also a Bus Master"



Cross-PIM Light Weight Threads

- Execution Model Assumption: Respond to many relatively short processing requests
  - Op to memory
  - "In the memory" Vector Operations
  - "In the memory" pointer chasing
  - Remote Method Invocation
  - Memory resident system introspection
- Communication via *parcels*:
  - Target address, method name, arguments
- Does not preclude supporting classical models
  - DSM across PIM nodes; Message passing

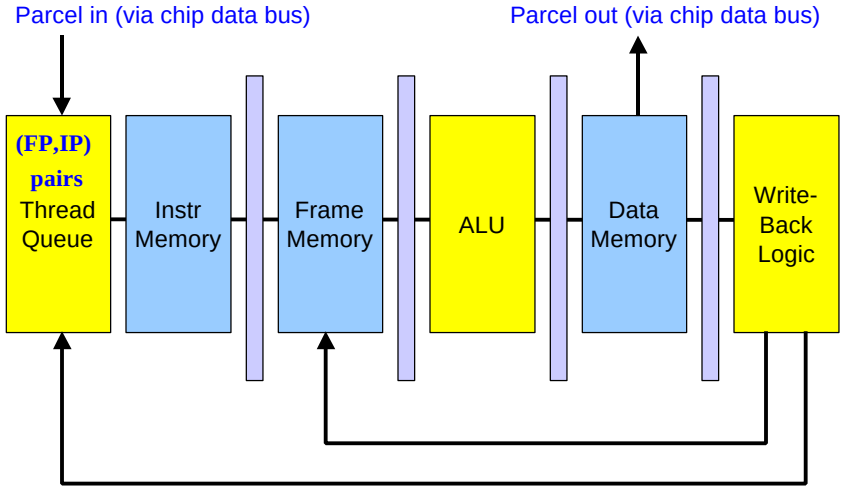
PIMLite: A First Prototype

- Quick, inexpensive prototype of multinode, multithreaded SRAM PIM chip
- Combined wide-word/multithreaded ISA
- Communication via hardware supported *parcels*:
- Status: Fab target April 7, 2003

PIMLite Machine State

- Each Thread: (FP, IP) pair
  - FP = Frame pointer to memory
  - IP = Instruction pointer
  - And that's it for hard state
- Frame: = length of PIM memory macro access
  - Today 8x256 = 2048 bits *IN MEMORY*
  - PIM Lite = only 128 bits
- Instructions manipulate contents of frame
  - Which can be managed in small frame cache

Pipeline



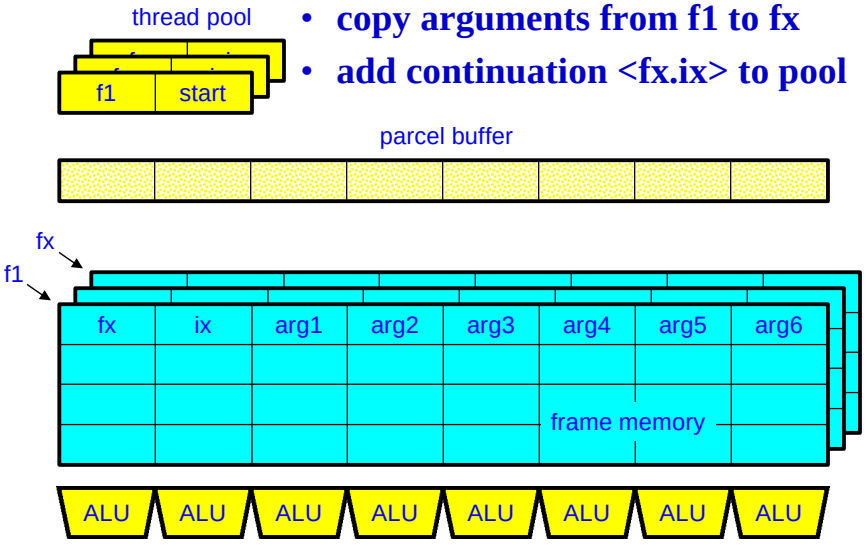
Multithreading Instructions

Instruction	Memory Operation	Conditions
fork $[N,q]_{ist}$, $[N,q]_{sem}$	$[N,q]_{sem} \leftarrow [N,q]_{sem} + 1$	<FP, $[N,q]_{ist}$> <FP, IP+1>
join $[N,q]_{sem}$	If $[N,q]_{sem} > 0$ $[N,q]_{sem} \leftarrow [N,q]_{sem} - 1$ else none	none <FP, IP+1>
sfob	none	none
sfalf $[N]_{src}$	copy $[N]_{src}$ to corresponding wide word in frame at $[N,0]_{src}$	$<[N,0]_{src}, [N,1]_{src}>$

ALU Instructions

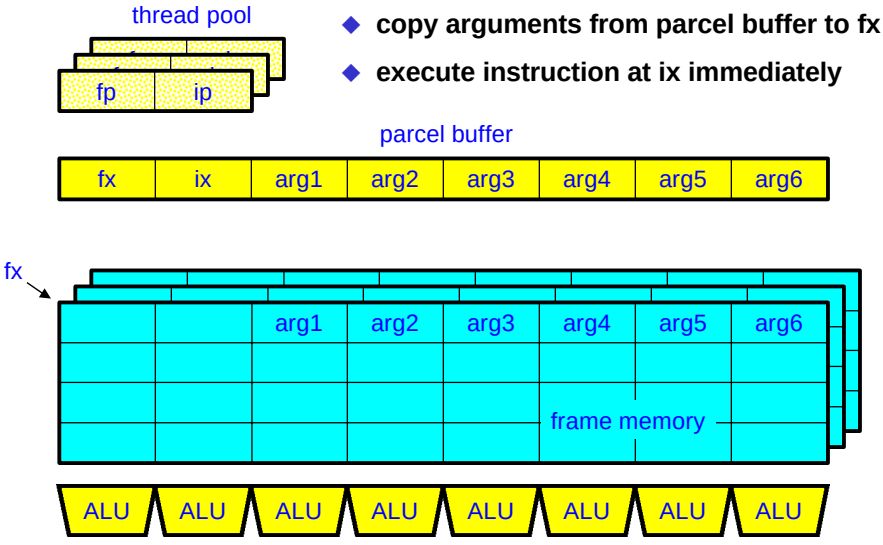
Instruction	Memory Operation	Conditions
vector-vector		
vop $[N]_{qst}$, $[N]_{src}$	$[N]_{qst} \leftarrow [N]_{qst}$ op $[N]_{src}$	<FP, IP+1>
permute $[N]_{qst}$, $[N]_{pos}$	permute elements of $[N]_{qst}$ by position vector $[N]_{pos}$	<FP, IP+1>
vector-scalar		
vsop $[N]_{qst}$, $[N,q]_{src}$	$[N]_{qst} \leftarrow [N]_{qst}$ op $[N,q]_{src}$	<FP, IP+1>
scalar-scalar		
op $[N,q]_{qst}$, $[N,q]_{src}$	$[N,q]_{qst} \leftarrow [N,q]_{qst}$ op $[N,q]_{src}$	<FP, IP+1>

Start New Thread



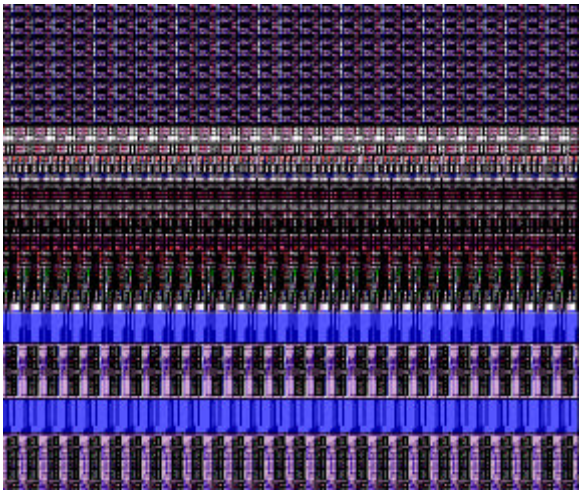
- copy arguments from f1 to fx
- add continuation <fx.ix> to pool

Fast Parcel Response



- copy arguments from parcel buffer to fx
- execute instruction at ix immediately

Pitch-Matched Layout



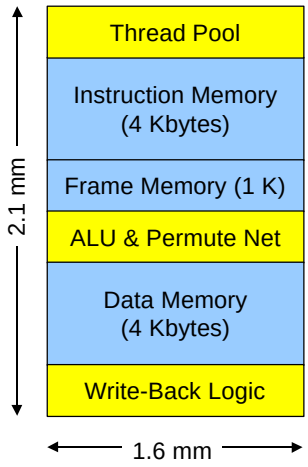
memory array

sense amplifiers

ALU logic

Chip Floorplan and Sizing

Single Node

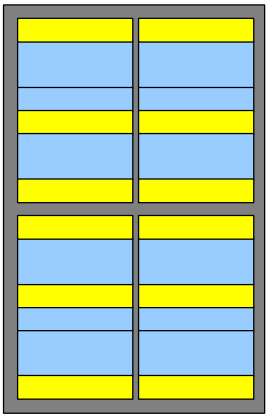


Each node
128 bits wide

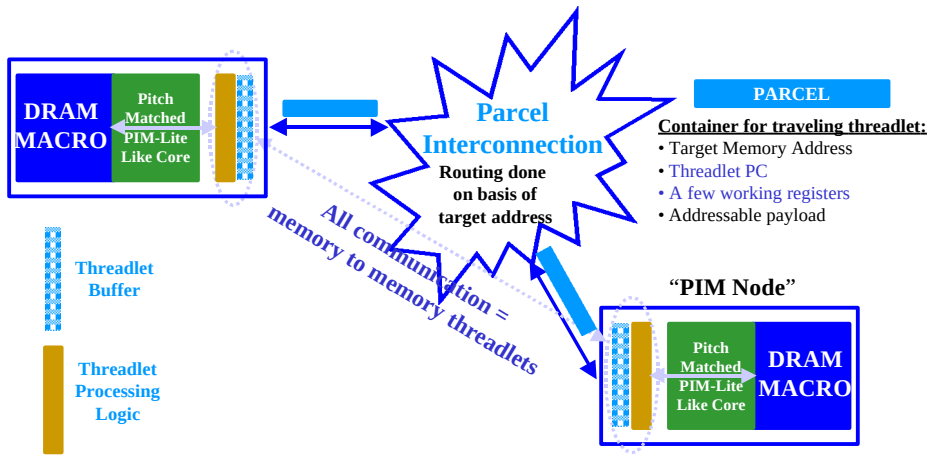
TSMC 0.18u

Transistor count
1 node:
500 K memory
60 K logic
Total:
2.25 M

4-Node Chip

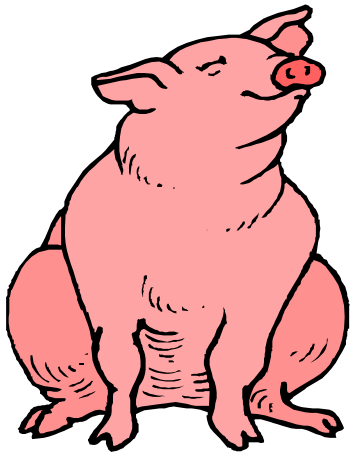


Ultra Light Weight "Traveling Thread"



Huge reduction in Transaction Count & Bandwidth Needs

The Future



Will We Design Like This?

Or This?