# CS184c:
# Computer Architecture
# [Parallel and Multithreaded]

Day 2:  April 5, 2001

Message Passing Mechanisms

## Today

- Message Driven Processor
- Mechanisms for Multiprocessing
- Engineering “Low cost” messaging

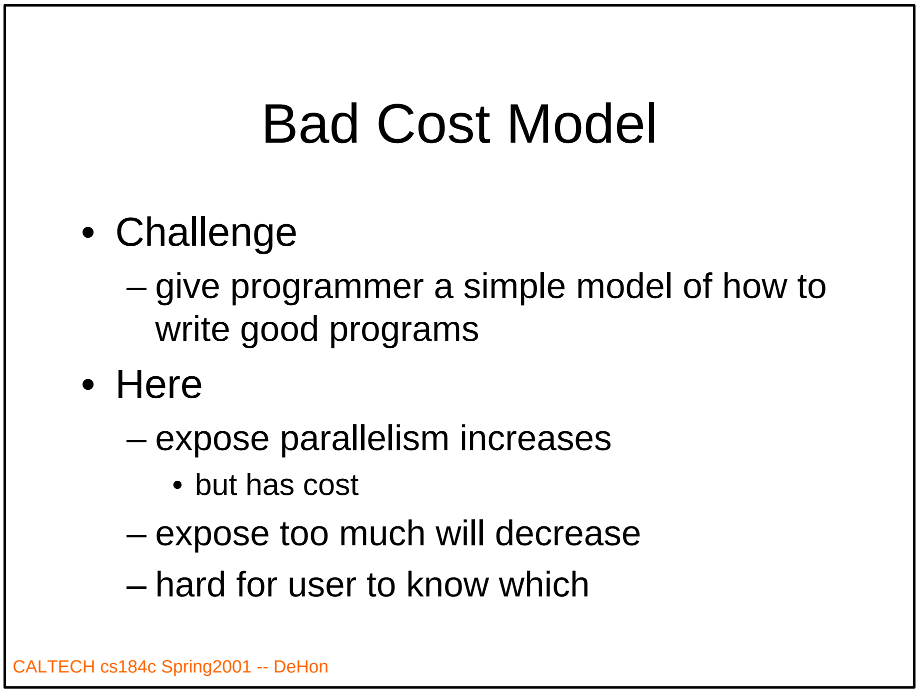
# Problem 1

- Messages take milliseconds
  - (1000s of cycles)

- Forces use of course-grained parallelism
  - Speedup = $T_{seq}/T_{mp} = c_{seq} \times N_p / c_{mp}$
  - $c_{seq} / c_{mp} \sim= t(comp) / (t(comm)+ t(comp))$
  - driven to make t(comp) >> t(comm)

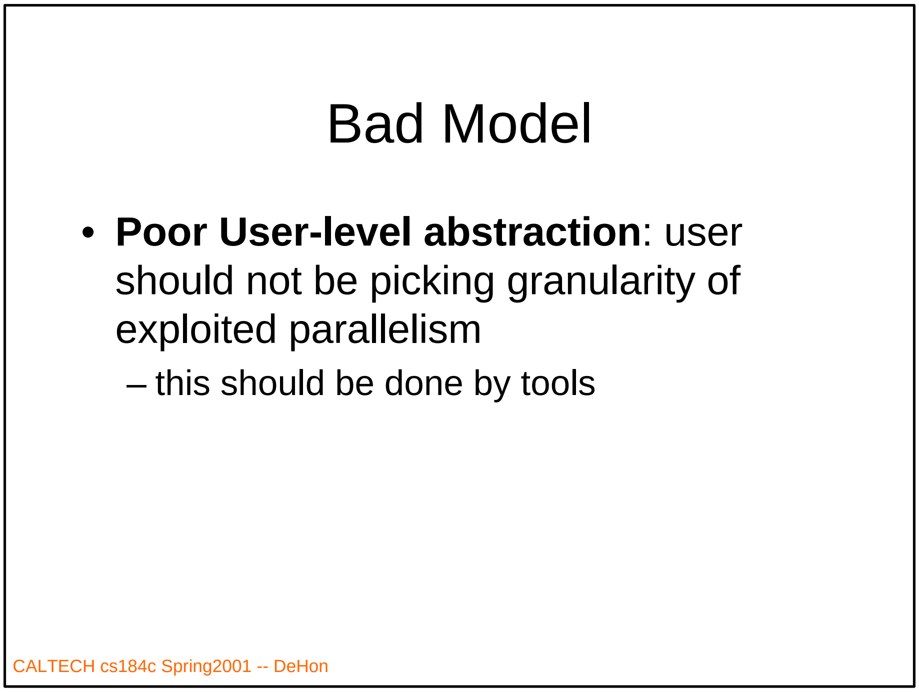
# Problem 2

- **Potential parallelism** is costly
  - additional communication cost is born even when sequentialized (same node)
- Process to process switch expensive
- Discourages exposing maximum parallelism
  - works against simple/scalable model



## Bad Cost Model

- Challenge
  - – give programmer a simple model of how to write good programs
- Here
  - – expose parallelism increases
    - but has cost
  - – expose too much will decrease
  - – hard for user to know which



## Bad Model

- **Poor User-level abstraction**: user should not be picking granularity of exploited parallelism
  - – this should be done by tools

# Cosmic Cube

- Used commodity hardware
  - off the shelf solution
  - components not engineered for parallel scenario
- Showed
  - could get benefit out of parallelism
  - exposed issues need to address to do it right
  - …why need to do something different

# Design for Parallelism

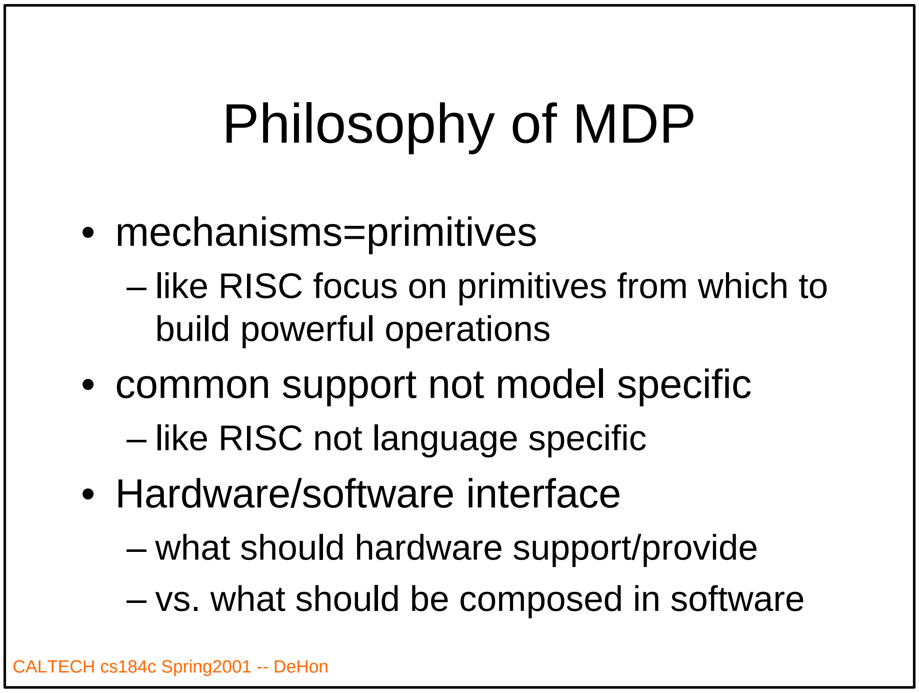
- To do it right
  - need to engineer for parallelism
- Optimize key common cases here
- Figuring out what goes in hardware vs. software

# Vision: MDP/Mosaic

- Single-chip, commodity building block
  - [today, tile to step and repeat on die]
  - contains all computing components
    - compute: sequential processor
    - interconnect in space: net interface + network
    - interconnect in time: memory
- Step-and-repeat competent uP
  - avoid diminishing returns trying to build monolithic processor

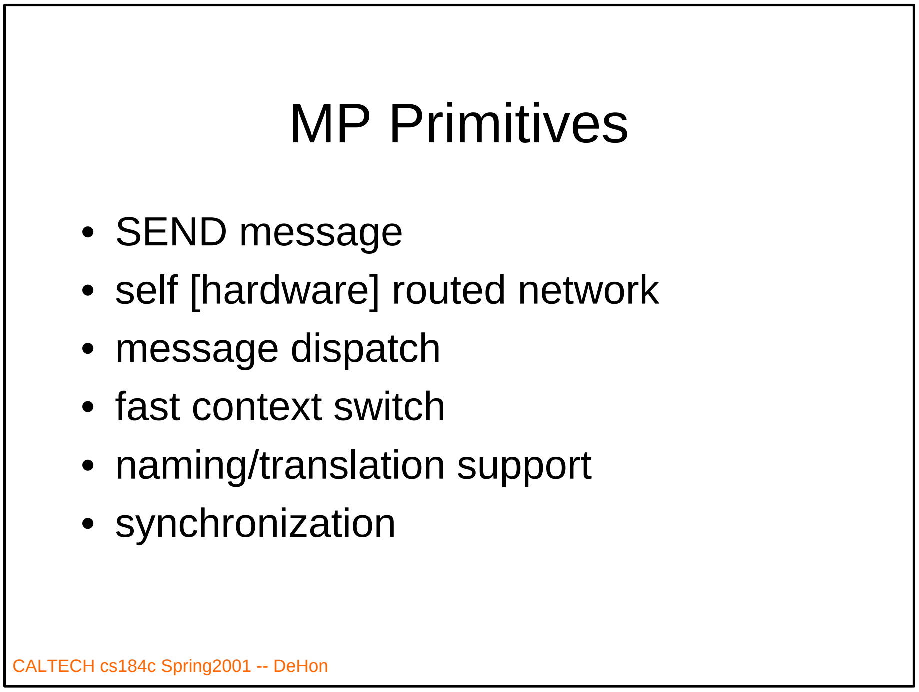
# Message Driven Processor

- “Mechanism” Driven Processor?
  - Study mechanisms needed for a parallel processing node
  - address problems saw in using existing
- View as low-level (hardware) model
  - underlies range of compute models
    - shared memory, dataflow, data parallel



## Philosophy of MDP

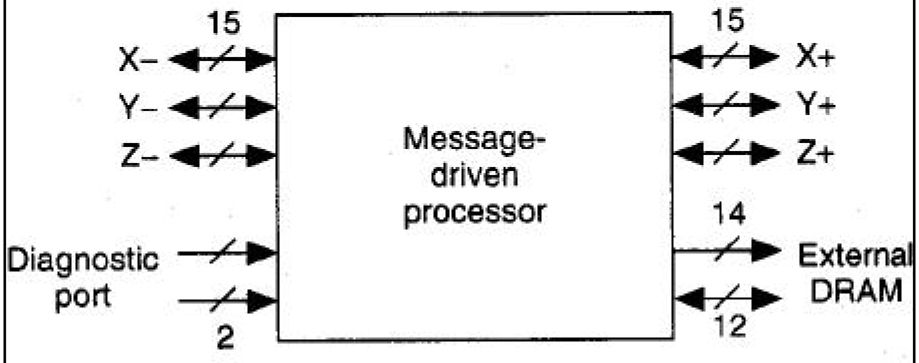
- mechanisms=primitives
  - like RISC focus on primitives from which to build powerful operations
- common support not model specific
  - like RISC not language specific
- Hardware/software interface
  - what should hardware support/provide
  - vs. what should be composed in software



## MP Primitives

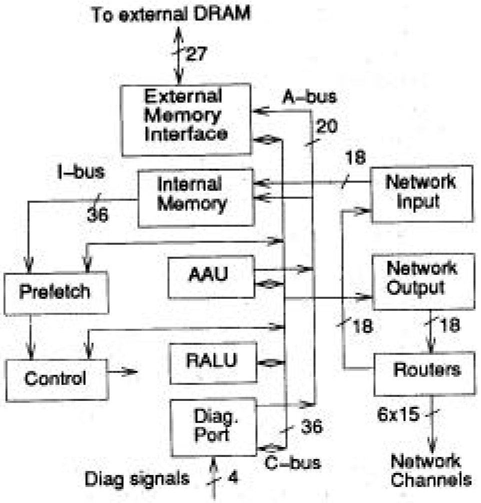
- SEND message
- self [hardware] routed network
- message dispatch
- fast context switch
- naming/translation support
- synchronization

# MDP Components

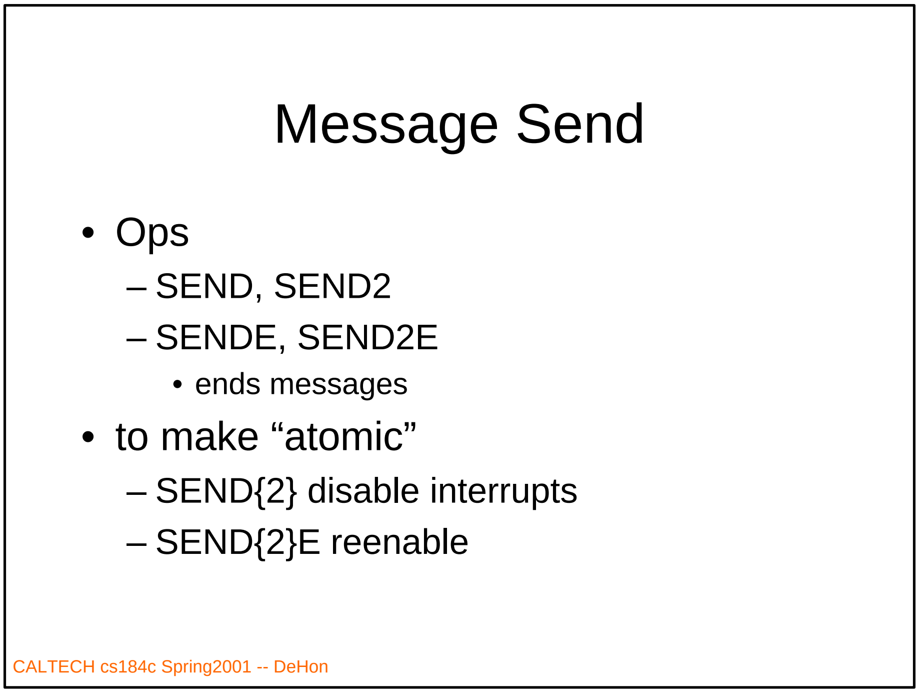


[Dally et. al.
IEEE Micro 4/92]

# MDP Organization

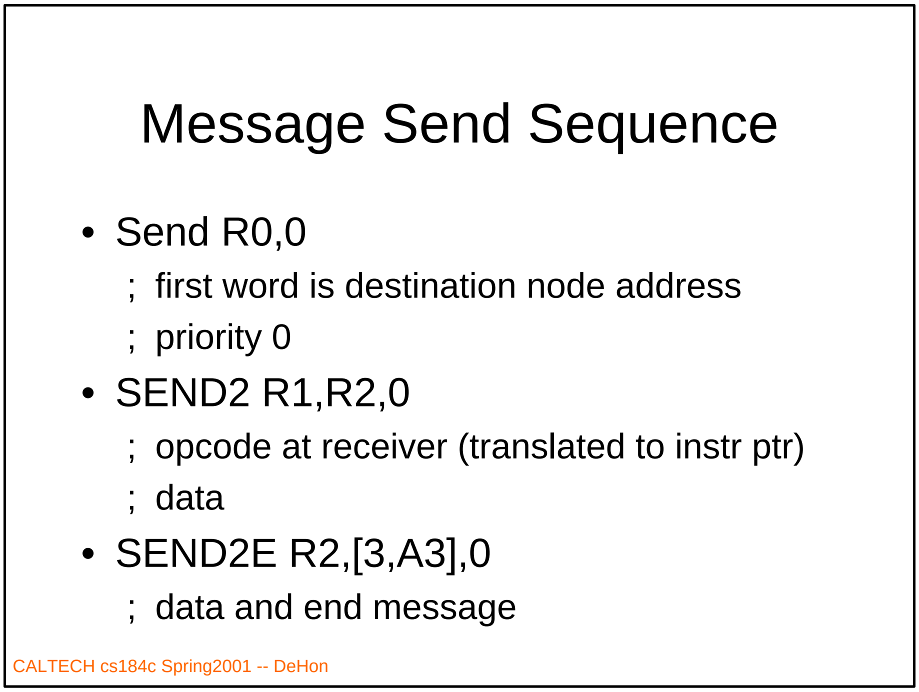


[Dally et. al.
ICCD'92]



## Message Send

- Ops
  - SEND, SEND2
  - SENDE, SEND2E
    - ends messages
- to make "atomic"
  - SEND{2} disable interrupts
  - SEND{2}E reenable



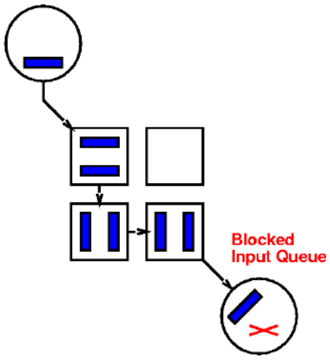
## Message Send Sequence

- Send R0,0
  - ; first word is destination node address
  - ; priority 0
- SEND2 R1,R2,0
  - ; opcode at receiver (translated to instr ptr)
  - ; data
- SEND2E R2,[3,A3],0
  - ; data and end message

## MDP Messages

- Few cycles to inject
- Not doing translation here
  - have to map from process to processor before can send
    - done by user code?
    - Trust user code?
  - Deliver to operation (address) on other end
    - receiver translates op to address
    - no protection

## Network



- 3D Mesh
  - wormhole
  - minimal buffering
  - dimension order routing
- hardware routed
  - orthogonal to node except enter/exit
  - contrast transputer
- messages can backup
  - all the way to sender

# Context Switch

- Why context switch expensive?
  - Exchange state (save/restore)
    - Registers
    - PC, etc.
    - TLB/cache...

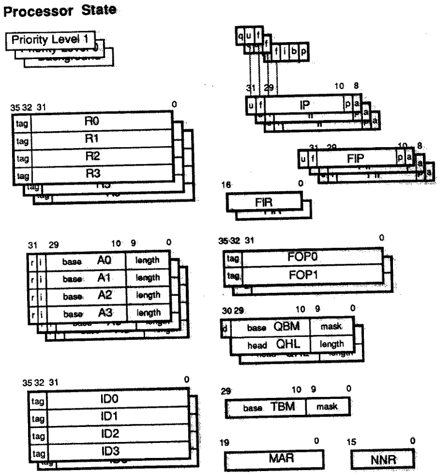
# Fast Context Switch

- General technique:
  - internal vs. external setup
- Machine Tool analogy
- Double-buffering

# Fast Context Switch

- Provide separate sets of Registers
  - trade space (more, large registers)
    - easier for MDP with small # of regs
  - for speed
- Don’t have to go through serialized load/store
- Probably also have to assure minimal/necessary handling code in fast memory

# MDP State



## Message Dispatch

- Incoming message queued by priority
- If higher priority than running (and interrupts enabled), will start running
  - few cycles to switch to “create” new task
- Terminated with suspend instruction
  - removes message from input queue

## Message Dispatch

- Idle MPD start running message after 3 cycles
  - set instruction pointer
  - create new message segment
  - A3 is message pointer

# Message Handler: CALL

- MOVE [1,A3],R0 ; get method ID
- XLATE R0,A0     ; translate to address
- LDIP     INITIAL_IP ; branch w/in seg

# Translation

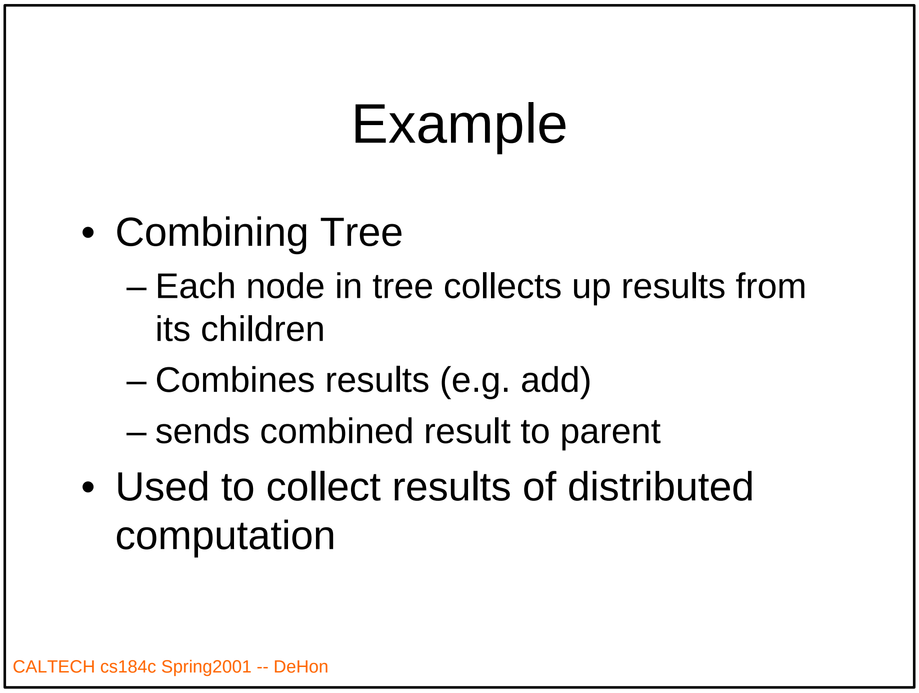
- XLATE
  - associative lookup
  - cache/TLB/mapping primitive
- ENTER
  - place an entry in associative table
  - may evict entry
- PROBE

# Translation

- XLATE used to map global ids to local memory
- could be used to map processes to processors?

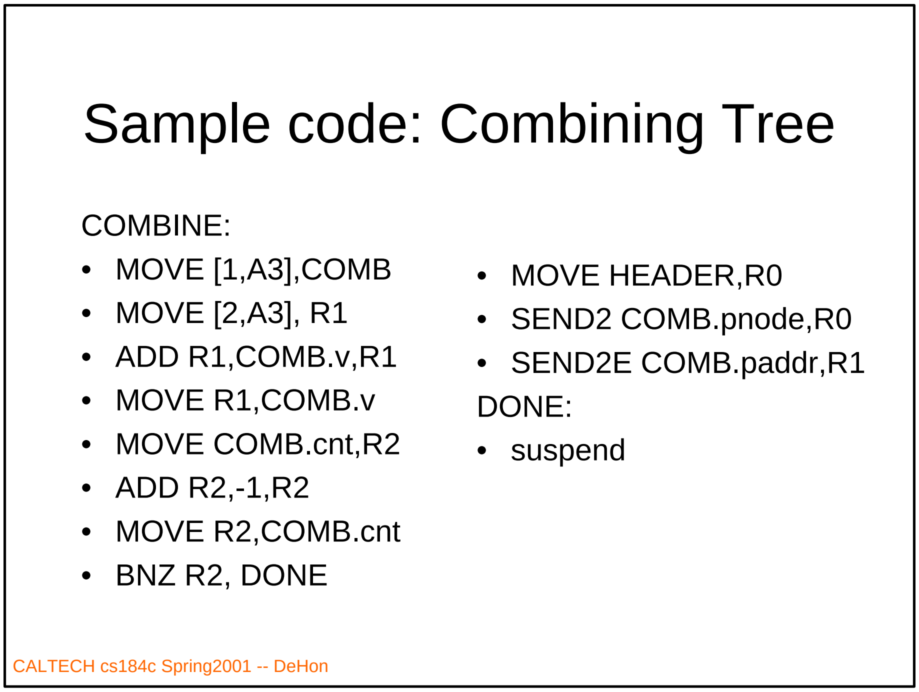
# Synchronization

- Future tags on data
  - [we’ll talk about futures later]



# Example

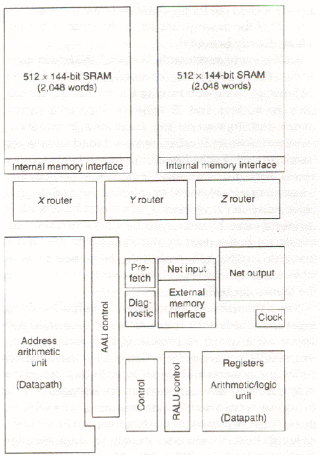
- Combining Tree
  - Each node in tree collects up results from its children
  - Combines results (e.g. add)
  - sends combined result to parent
- Used to collect results of distributed computation

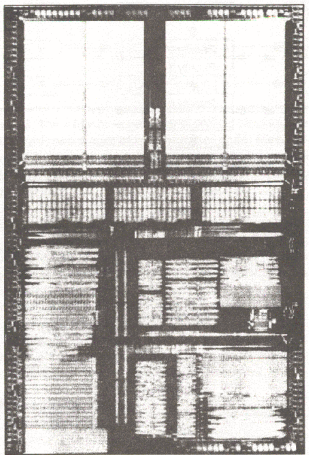


# Sample code: Combining Tree

```
COMBINE:
•  MOVE [1,A3],COMB
•  MOVE [2,A3], R1
•  ADD R1,COMB.v,R1
•  MOVE R1,COMB.v
•  MOVE COMB.cnt,R2
•  ADD R2,-1,R2
•  MOVE R2,COMB.cnt
•  BNZ R2, DONE
•  MOVE HEADER,R0
•  SEND2 COMB.pnode,R0
•  SEND2E COMB.paddr,R1
DONE:
•  suspend
```

# MDP Area





# MDP Area

- Memory ~50%
- Processor ~33%
- Net ~10%

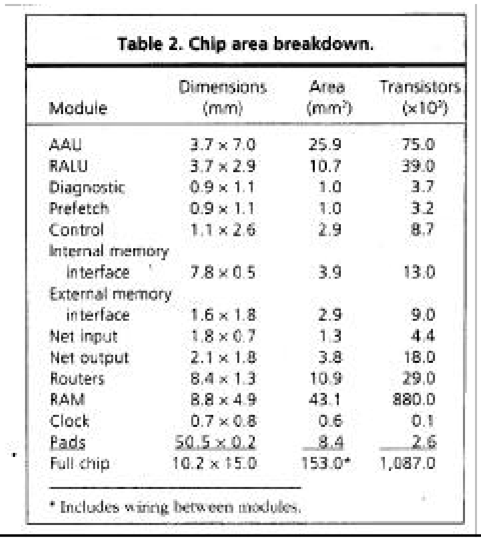
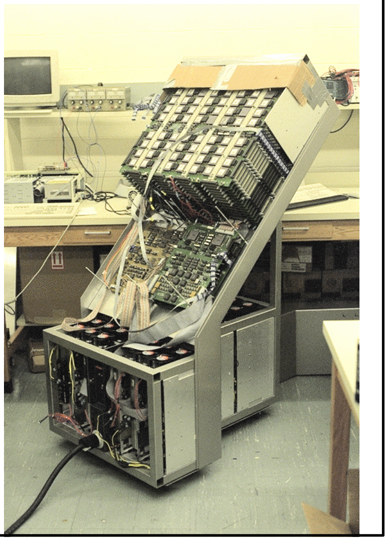


**Table 2. Chip area breakdown.**

| Module | Dimensions (mm) | Area ($mm^2$) | Transistors ($\times 10^3$) |
|---|---|---|---|
| AAU | 3.7 × 7.0 | 25.9 | 75.0 |
| RALU | 3.7 × 2.9 | 10.7 | 39.0 |
| Diagnostic | 0.9 × 1.1 | 1.0 | 3.7 |
| Prefetch | 0.9 × 1.1 | 1.0 | 3.2 |
| Control | 1.1 × 2.6 | 2.9 | 8.7 |
| Internal memory interface | 7.8 × 0.5 | 3.9 | 13.0 |
| External memory interface | 1.6 × 1.8 | 2.9 | 9.0 |
| Net input | 1.8 × 0.7 | 1.3 | 4.4 |
| Net output | 2.1 × 1.8 | 3.8 | 18.0 |
| Routers | 8.4 × 1.3 | 10.9 | 29.0 |
| RAM | 8.8 × 4.9 | 43.1 | 880.0 |
| Clock | 0.7 × 0.8 | 0.6 | 0.1 |
| Pads | 50.5 × 0.2 | 8.4 | 2.6 |
| Full chip | 10.2 × 15.0 | 153.0* | 1,087.0 |

* Includes wiring between modules.

# J-Machine

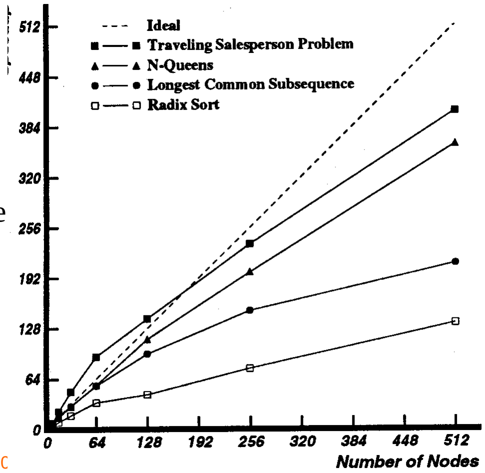


# Performance

- Base communication: 1μs node to node
- Empty ping: 3-7μs round trip
  - depends on distance
  - 43 cycles round trip for node pinging self
- MDP 12.5 MIPs
  - 2 MIPs when fetching instructions from external memory

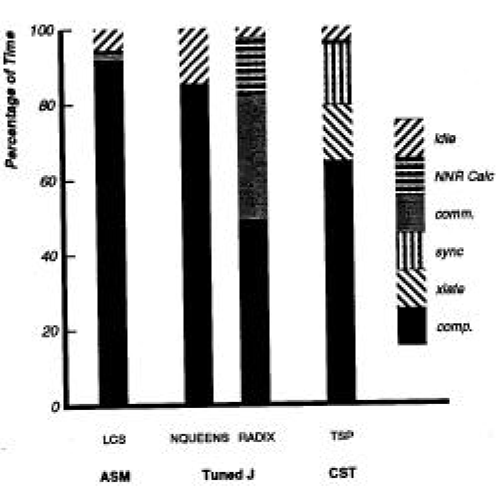
# Performance Results

Note:
all relative to MDP;
not show slowdown to parallel code and MDP.



[Noakes, Wallach Dally ISCA'93]

# Time Decomposition



[Noakes, Wallach Dally ISCA'93]

# Other Lessons

- “Mechanisms” important for uniprocessor performance important here as well
  - hardware memory hierarchy management
    - caching, TLB
  - floating point hardware
  - large register set

# Observation

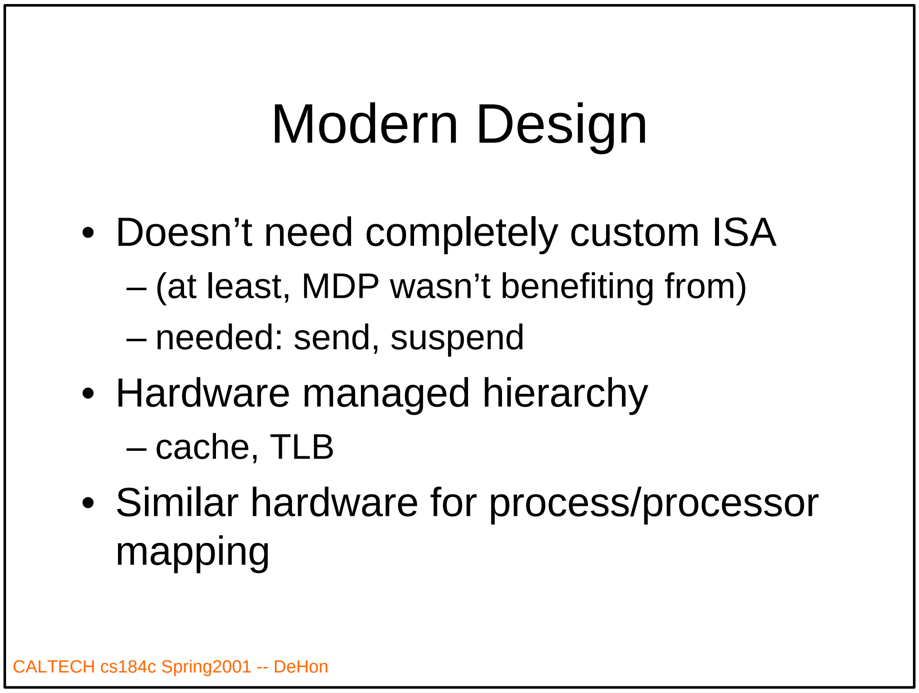
- Anything with a different programming model is hard to sell
- …especially if some component of your machine is **worse** than conventional alternatives
  - communication in Cosmic Cube
  - scalar (esp. FP) performance in J-Machine

# Non-Lessons

- Balance
  - – network overpowered for node
    - 3× speed of external memory
- Network
  - – dimension order routing
  - – “efficiency” of wire utilization
  - – [will return to in week 8]

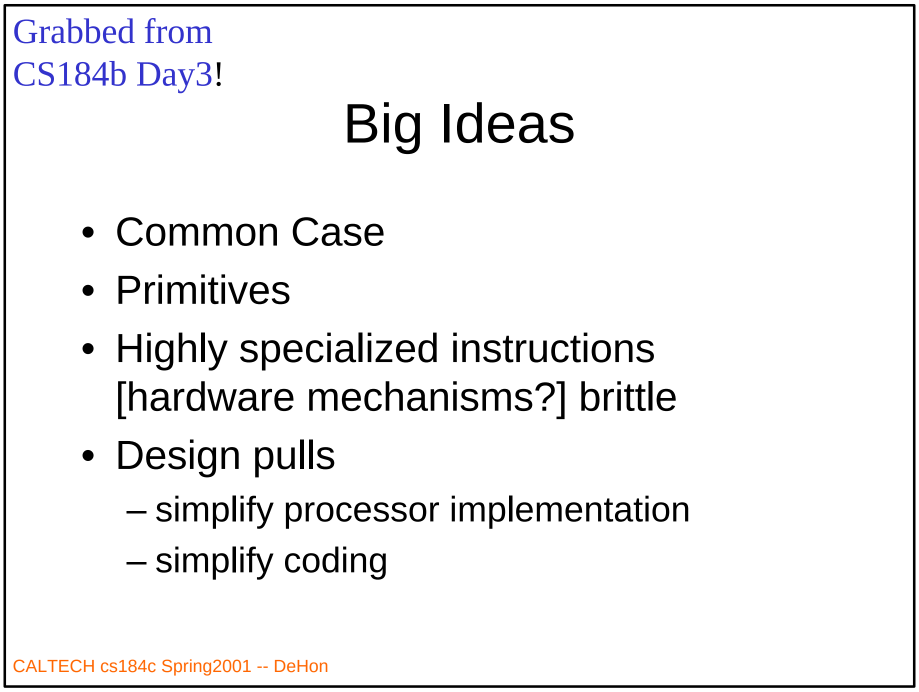
# Follow ons...

- M-Machine (research)
- Cray T3D
- ASCII Red



## Modern Design

- Doesn’t need completely custom ISA
  - (at least, MDP wasn’t benefiting from)
  - needed: send, suspend
- Hardware managed hierarchy
  - cache, TLB
- Similar hardware for process/processor mapping



Grabbed from
CS184b Day3!

## Big Ideas

- Common Case
- Primitives
- Highly specialized instructions [hardware mechanisms?] brittle
- Design pulls
  - simplify processor implementation
  - simplify coding

# Big Ideas

- Compiler: fill in gap between user and hardware architecture
  - good idea, not being exploited here
- Need different/additional primitives for handling parallel cooperation efficiently
  - communication
  - cheap process virtualization