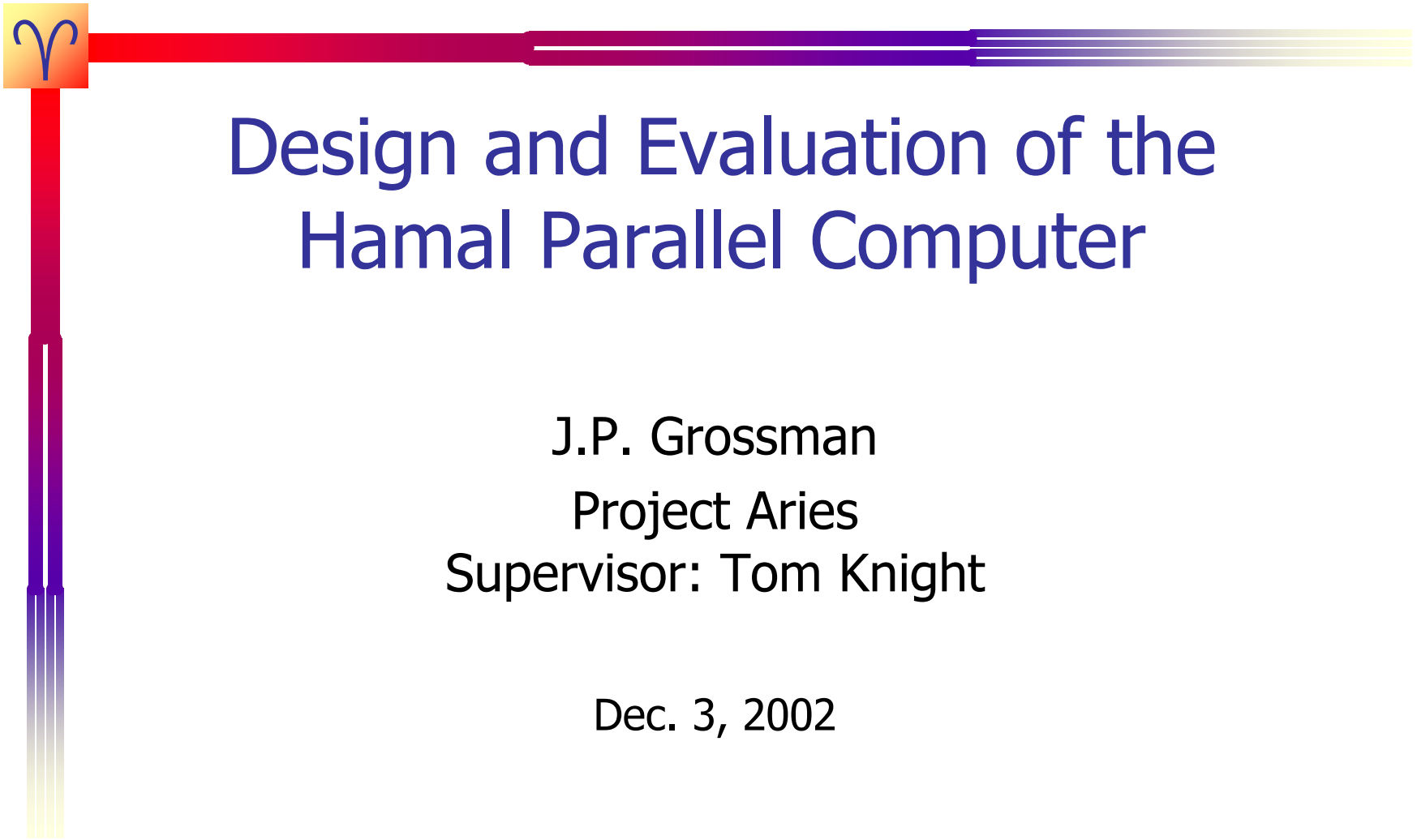


Design and Evaluation of the Hamal Parallel Computer

J.P. Grossman

Project Aries

Supervisor: Tom Knight

Dec. 3, 2002

Motivation

- Million node general-purpose machine
  - Scalable memory system
  - Support for massive multithreading
  - Discarding Network
- Billion transistor era
- Embedded DRAM

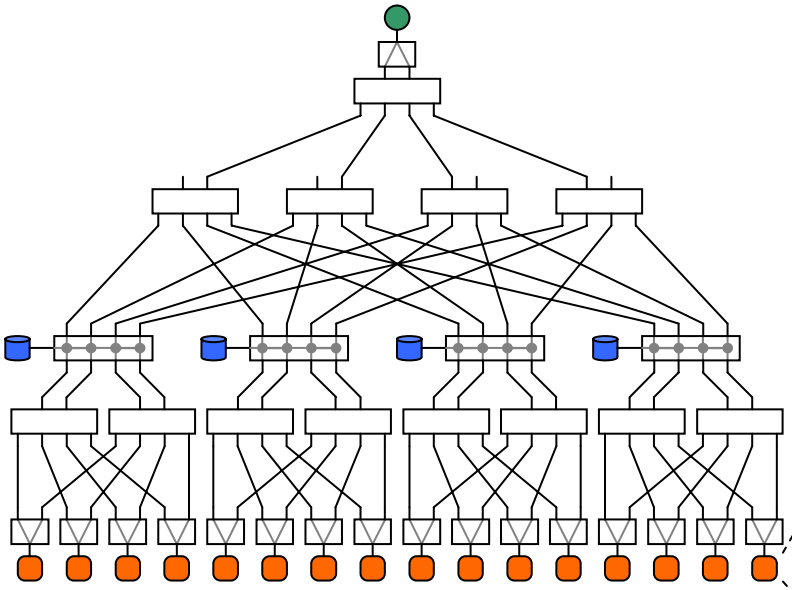
Talk Outline

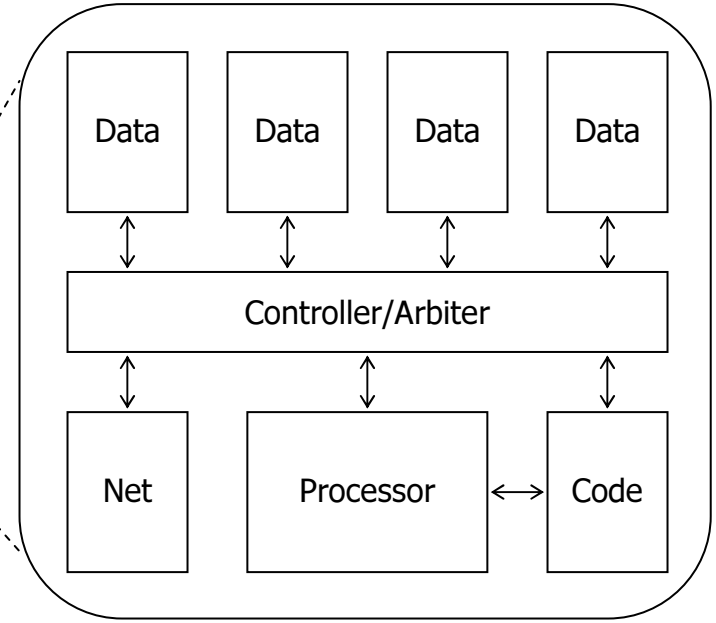
- Overview of Hamal
- The Hamal memory system
- Thread management and synchronization
- Fault-tolerant messaging protocol

Hamal - Overview

- Distributed shared memory machine
- Multiple processor-memory nodes per die
- Fat tree interconnect
- Split-phase memory operations
- Memory consistency implemented in software
- No data caches
- Complete system cycle-accurate simulator

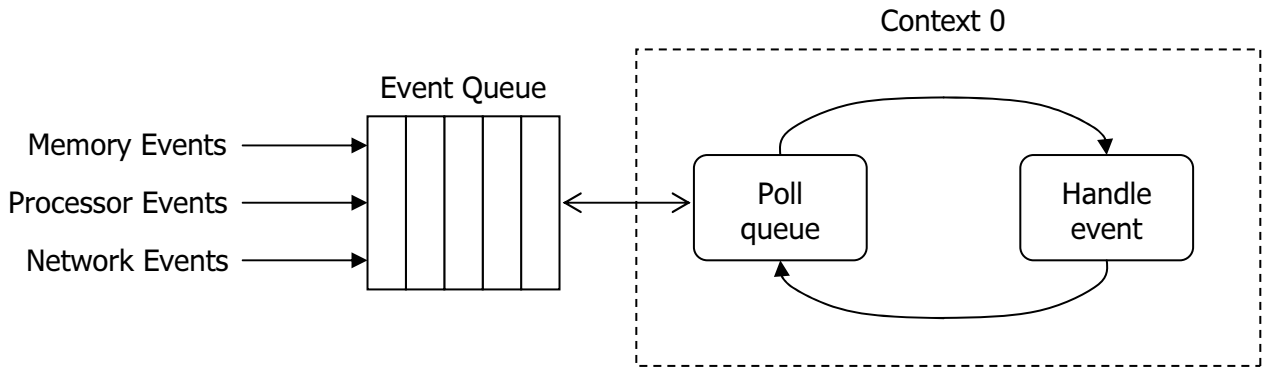
Hamal - Overview





The Hamal Processor

- 128-bit VLIW multithreaded (8 contexts)
- No register renaming, branch prediction, speculative execution, etc.
- Event-driven microkernel runs in context 0



Talk Outline

- ✓ Overview of Hamal
- The Hamal memory system
- Threads and synchronization
- Fault-tolerant messaging protocol

Capabilities

- 128 bit pointers with embedded hardware-enforced permissions and bounds
  - 64 address bits, 64 capability bits
- Single virtual address space
  - Reduces state associated with a process
  - Easy sharing of data
- Intra-process protection
- Object-based protection
- Simple lazy page allocation

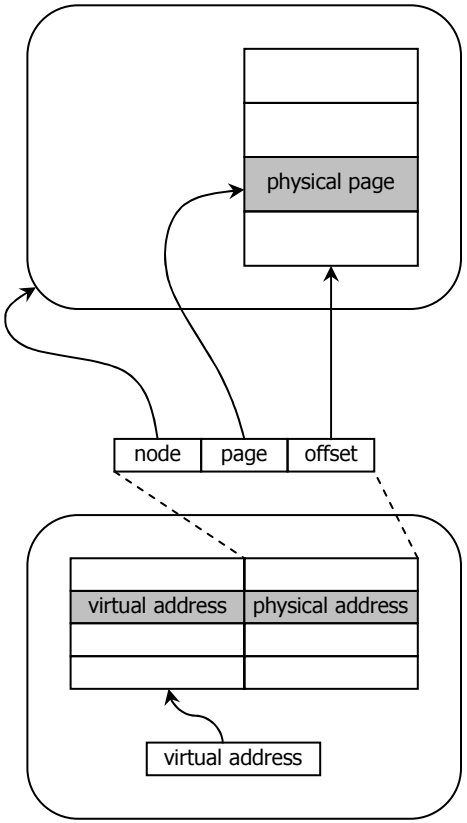
A Haiku

Capabilities!

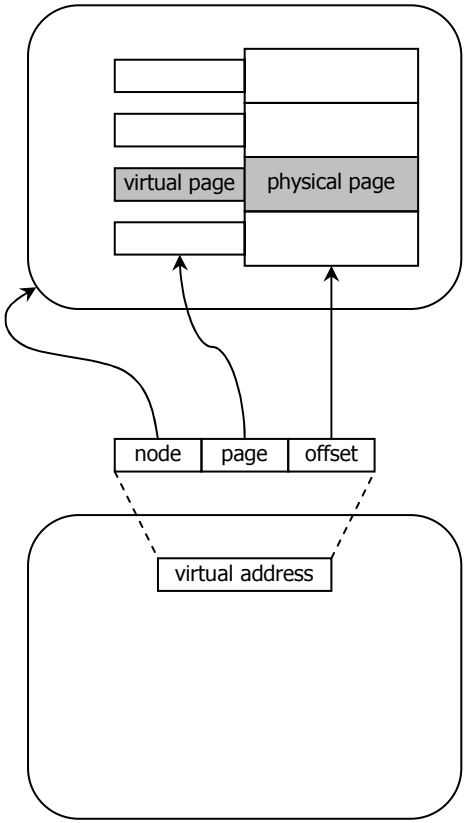
It is no longer a sin

to program in C

Virtual Memory



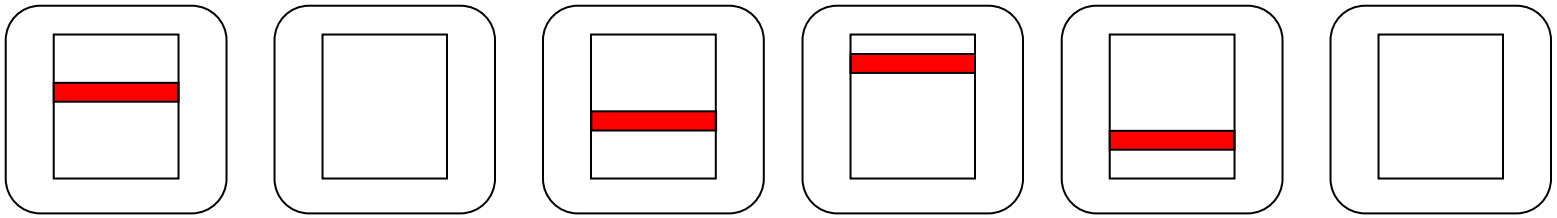
TLB



Hardware Page Tables

Distributed Objects

- Hamal implements *Sparsely Faceted Arrays* [Brown02]
- Conceptually allocate same segment on all nodes, but actual facets are lazily allocated
- Network interface translates between global segment IDs and local facets



Talk Outline

- ✓ Overview of Hamal
- ✓ The Hamal memory system
- Threads and synchronization
- Fault-tolerant messaging protocol

Motivation

- Run time for a problem of size m on N nodes:

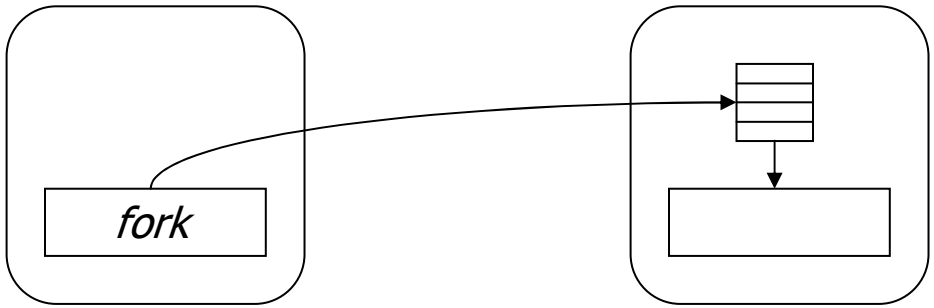
$$t = C_1 \frac{m}{N} + C_2 \log(N)$$

- Optimal run time for fixed m:

$$C_2 = C_1 \frac{m}{N} \quad \Rightarrow \quad t = C_2 \left(1 + \log\left(\frac{C_1 m}{C_2} \right) \right)$$

Thread Creation

- *fork* instruction specifies:
  - Starting address
  - Destination node
  - Set of registers to copy to child
- Each node contains a hardware *fork queue*
- Queue is serviced by microkernel



Register Dribbling

- Each thread has a *swap page* in memory
- Threads are loaded/unloaded in the background on unused memory cycles (*Register dribbling* - [Soundararajan92])
  - Reduces *overhead* of thread swapping
- Least recently issued (LRI) context constantly dribbles
  - Reduces *latency* of thread swapping

Thread Suspension

- When should a blocked thread be suspended?
- Two part strategy:
  1. Wait until
     a) No context can issue
     b) The LRI context is clean
  2. Generate a *stall* event and allow the microkernel to decide if the thread should be suspended

Register-Based Synchronization

- *join capabilities* allow one thread to write directly to another thread's registers
- Three instructions: *jcap*, *busy*, and *join*

Parent Thread

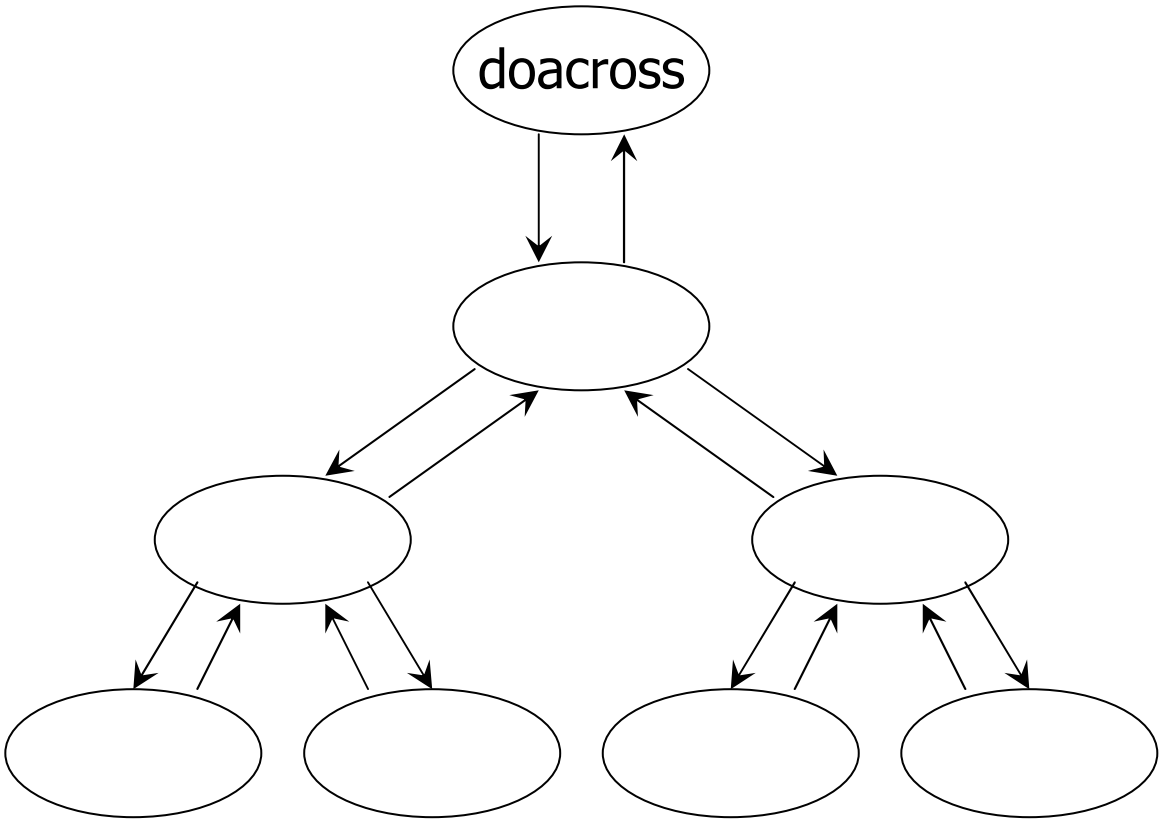
```
r0 = jcap r1
r1 = busy
fork _child, {r0}
r1 = and r1, r1
```

Child Thread

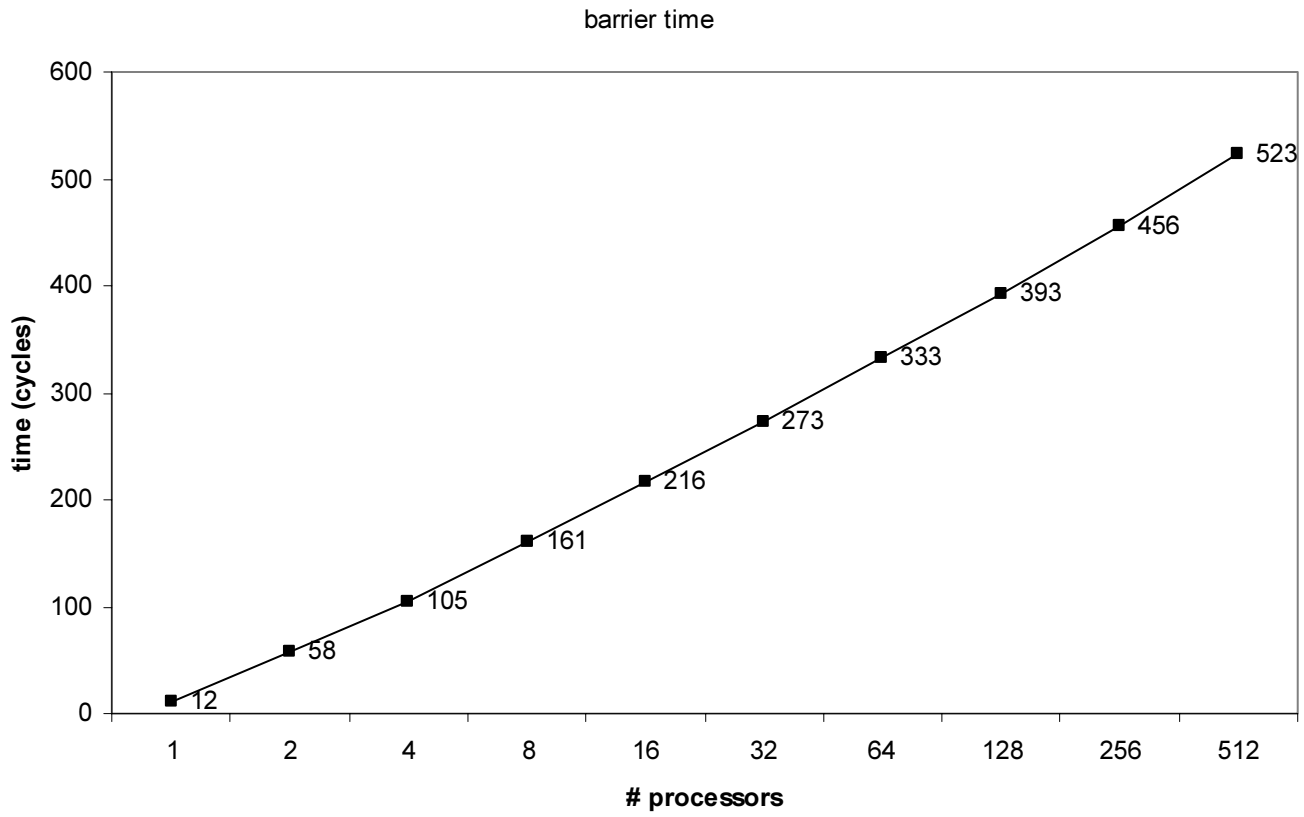
```
_child:

<computation>
join r0, 0
```

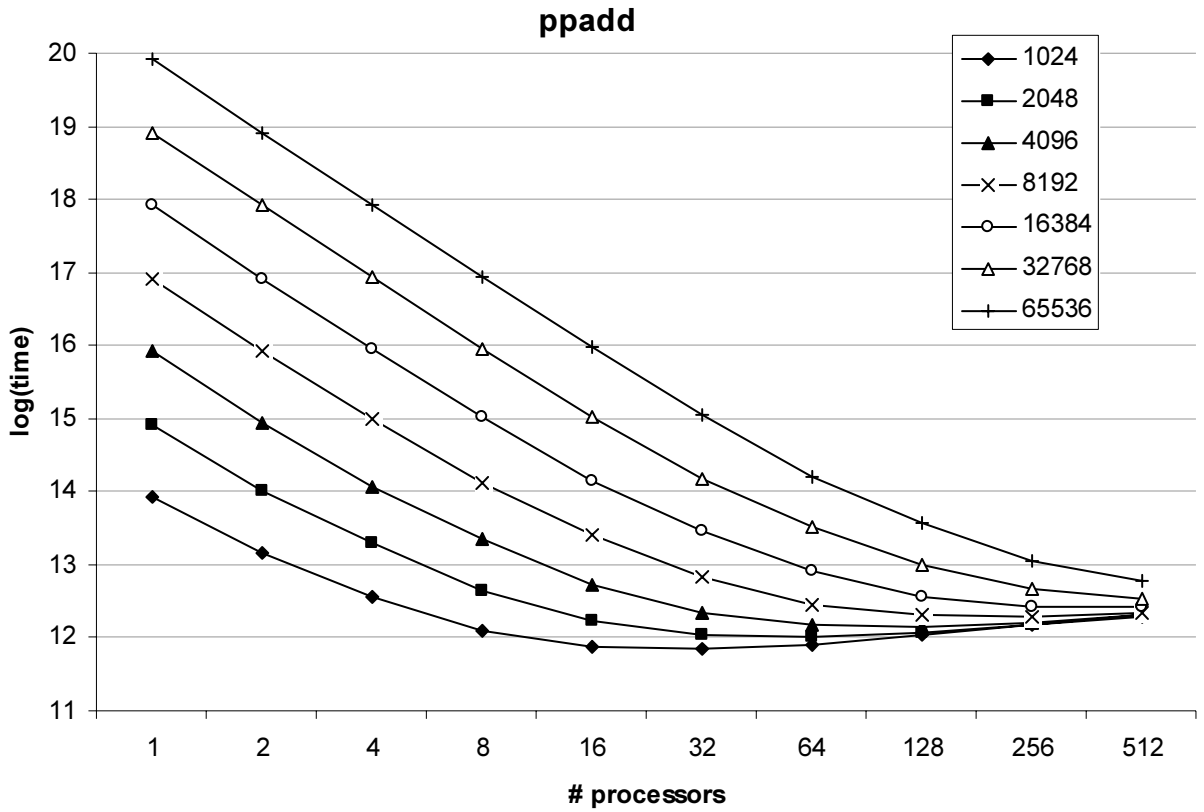
Example - Barrier



Barrier Times



Benchmark - *ppadd*



UV Trap Bits

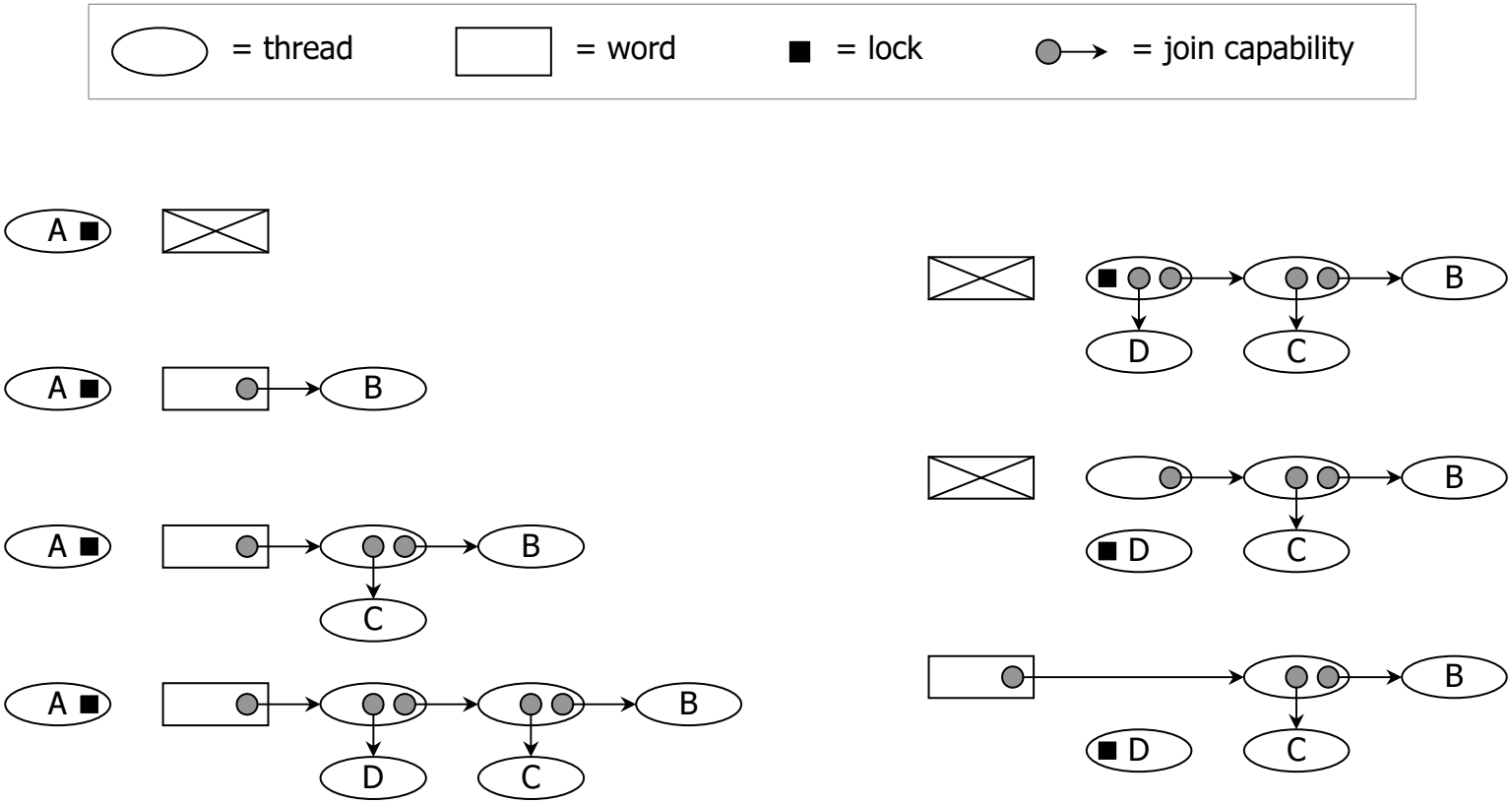
- Each memory word is tagged with two user trap bits (U, V)
- Each memory operation may optionally:
  - Trap on U high, U low, V high, V low
  - Modify U, V if successful
- Traps generate events which are handled by the microkernel on the node containing the memory word

Example – Word Locking

U	V	Meaning
0	0	available
1	0	unavailable
0	1	trap
1	1	busy

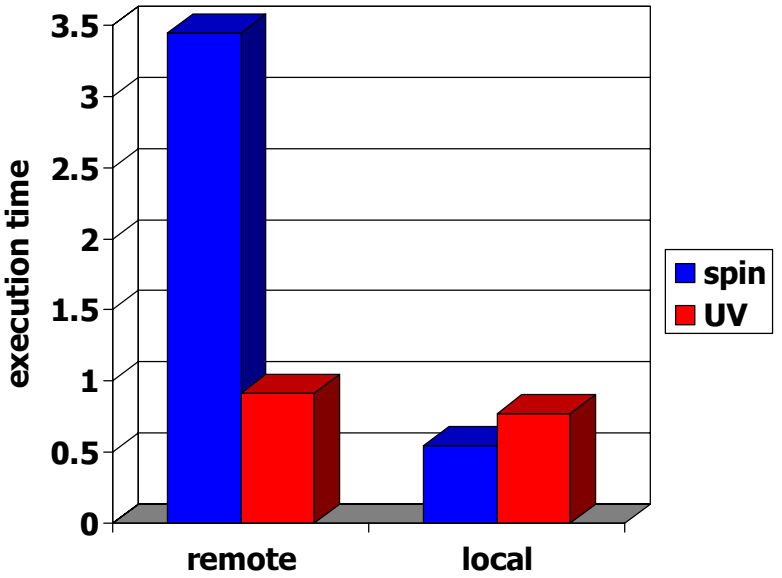
- Aqcuire:
  - load, trap on U high or V high, set U
- Release:
  - store, trap on V high, clear U

Example – Word Locking



Benchmark – *wordcount*

- Frequency count of words in [Brown02]
- Distributed hash table used to store counts
- *remote version*: access hash table remotely
- *local version*: create a thread on target node



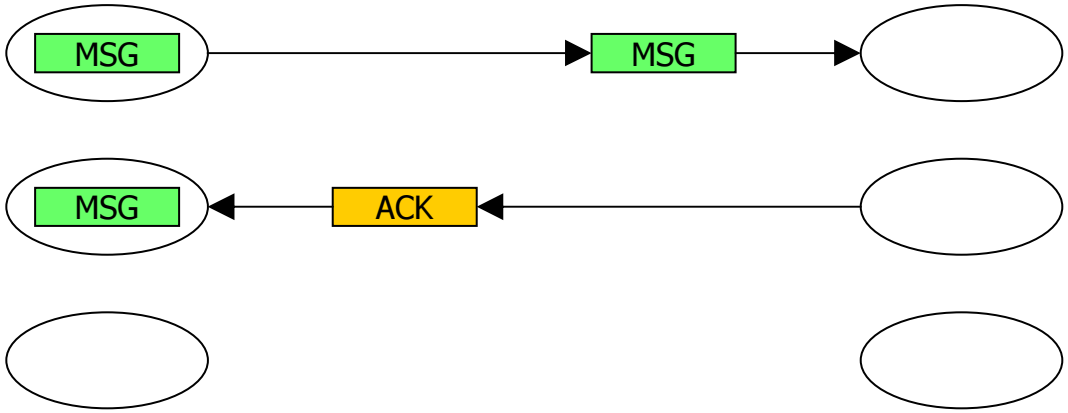
Talk Outline

- ✓ Overview of Hamal
- ✓ The Hamal memory system
- ✓ Threads and synchronization
- Fault-tolerant messaging protocol

Motivation

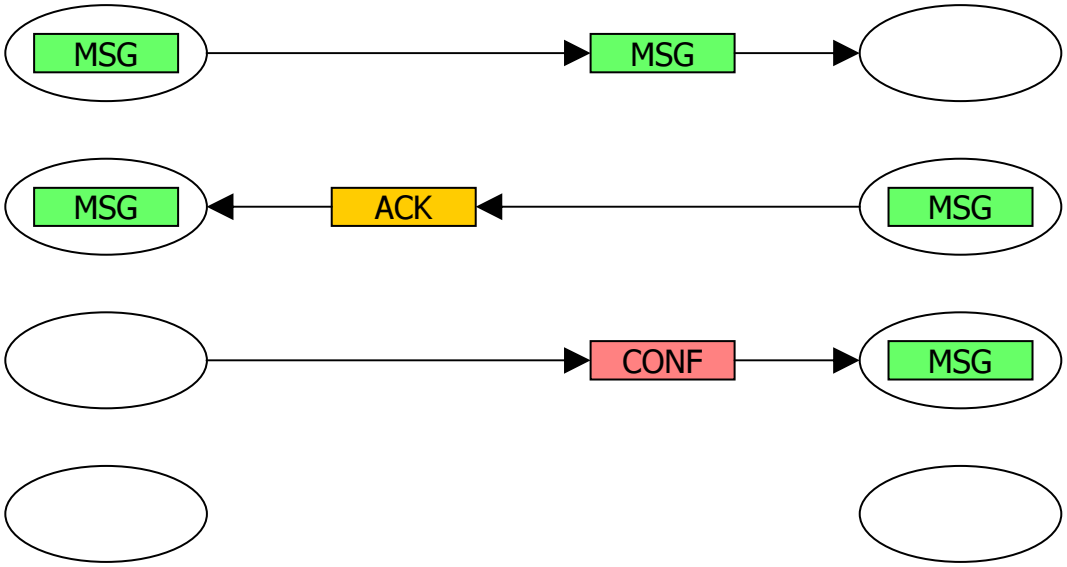
Discarding	vs.	Non-Discarding
Internet	Examples	Most || Computers
	Performance	✓
✓	Simplicity	✗
✓	Reliability	✗

Fault Tolerant Messaging



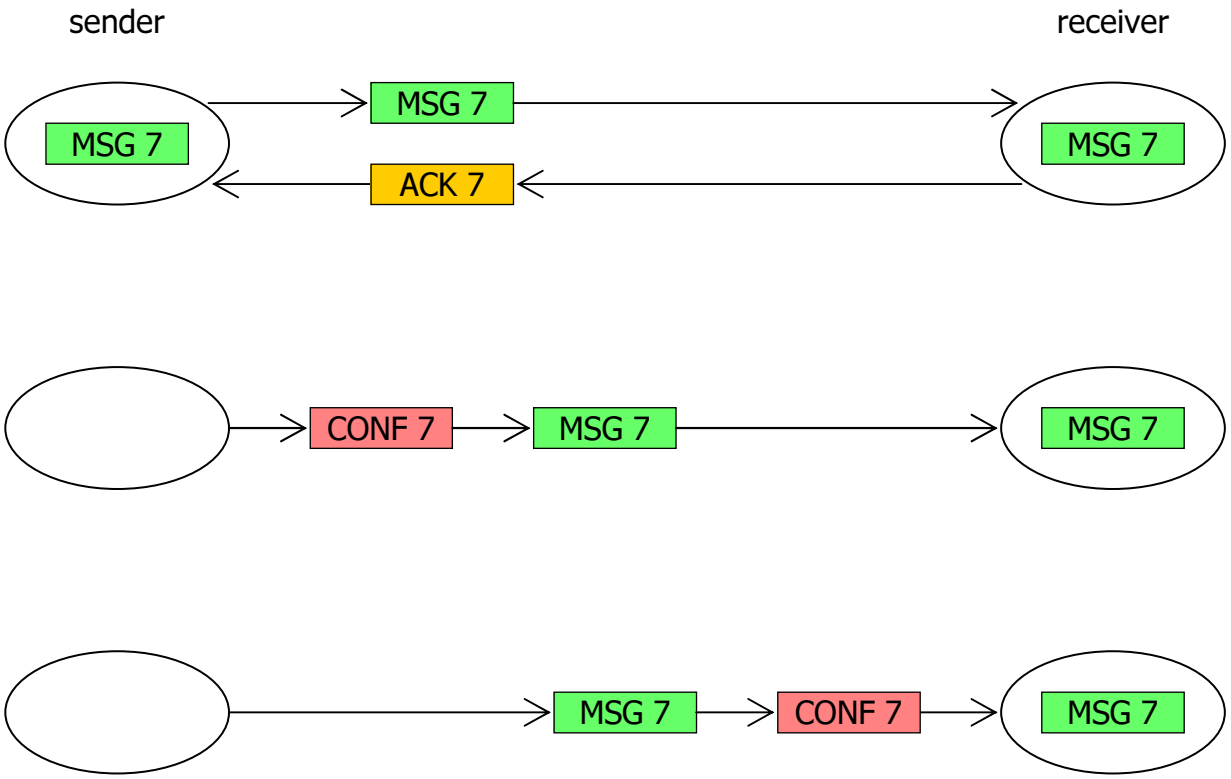
- Idempotence?
- Sequence numbers (e.g. TCP)
  - 2^{20} nodes, 32 bits $\Rightarrow$ 8MB/node

Idempotent Messaging Protocol



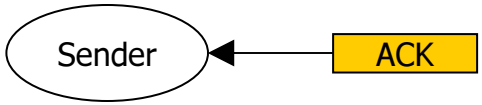
- CONF: No more messages will be sent
  - [Brown01]

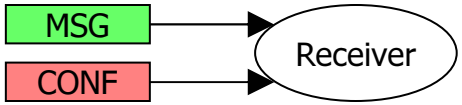
Out of Order Packets



Message Identification

- Sender generates message ID
- All packets contain source node and msg. ID





- ID identifies MSG
- source node gives destination for CONF

- (source node, ID) identifies MSG

Can We Reduce Overhead?

- ACK/CONF packets:

type	dest	source	message ID

- Two ideas to reduce size:
  1. Use short (4-8 bit) MSG ID
  2. Receiver assigns short secondary ID

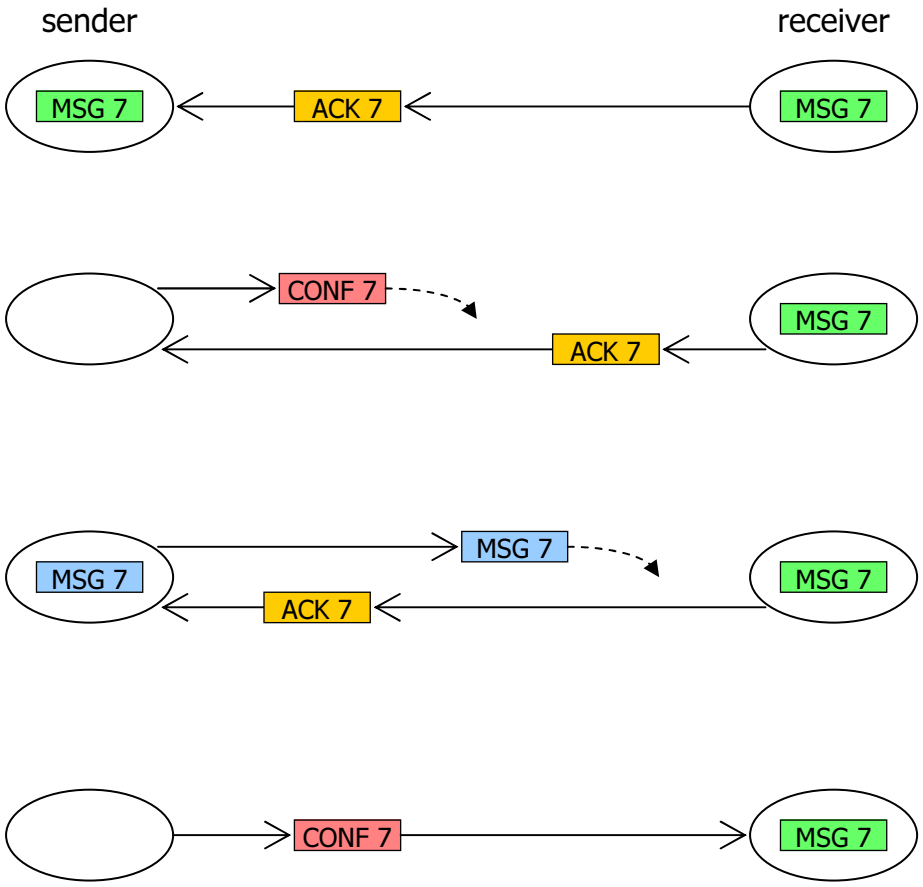
ACK:

type	dest	source	message ID	ID2

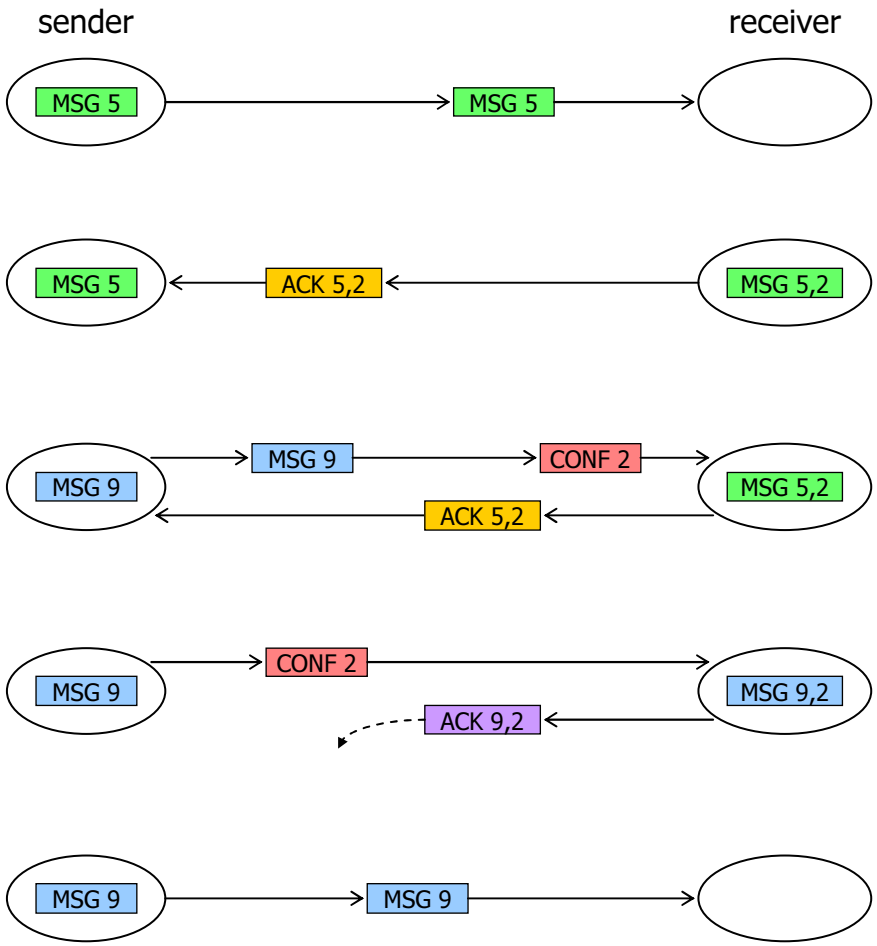
CONF:

type	dest	ID2

Failure of Short IDs



Secondary IDs



How Many Bits Is Enough?

- Can't reuse an ID if it's still in the system
- But receivers can remember an ID for arbitrarily long periods of time
- Solution:
  - use 48 bit IDs
  - flush the network every 4-12 months when a node runs out of IDs

Experimental Results

- Trace driven simulation of 4 microbenchmarks on 4 topologies
- Linear backoff for packet retransmission
- Small send tables (8 entries)
- Larger receive tables (64 entries)
- Buffer ~4 flits at each network node

Summary

- Scalable memory system
  - Capabilities → single shared virtual address space
  - Hardware page tables, sparsely faceted arrays
- Low overhead for parallel programs
  - Efficient thread management
  - Register-based synchronization
  - UV Trap bits
- Discarding network
  - Lightweight fault-tolerant messaging protocol

Conclusion

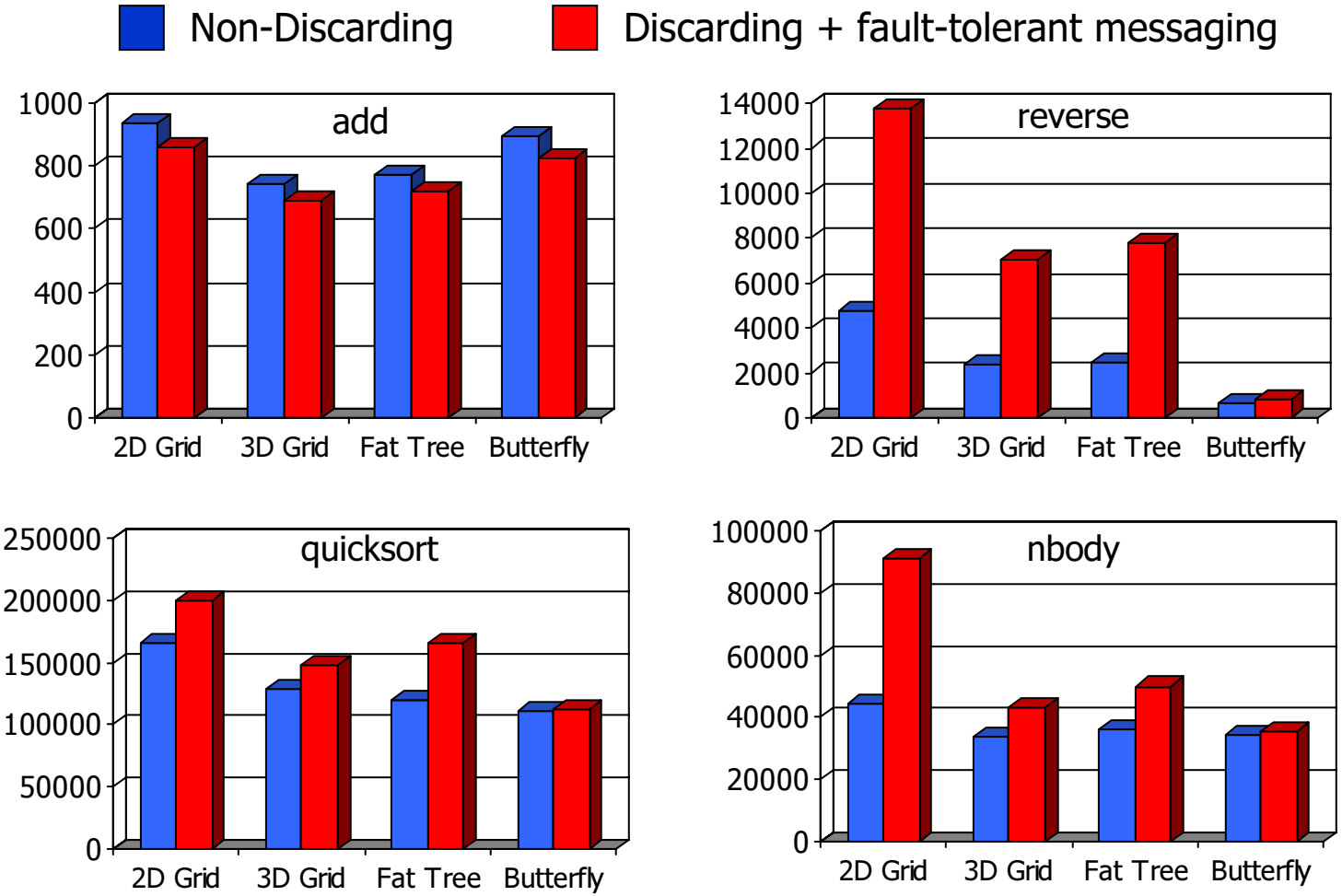
Yesterday – ENIAC

Today – P4

Soon – 1M nodes

Tomorrow – ???

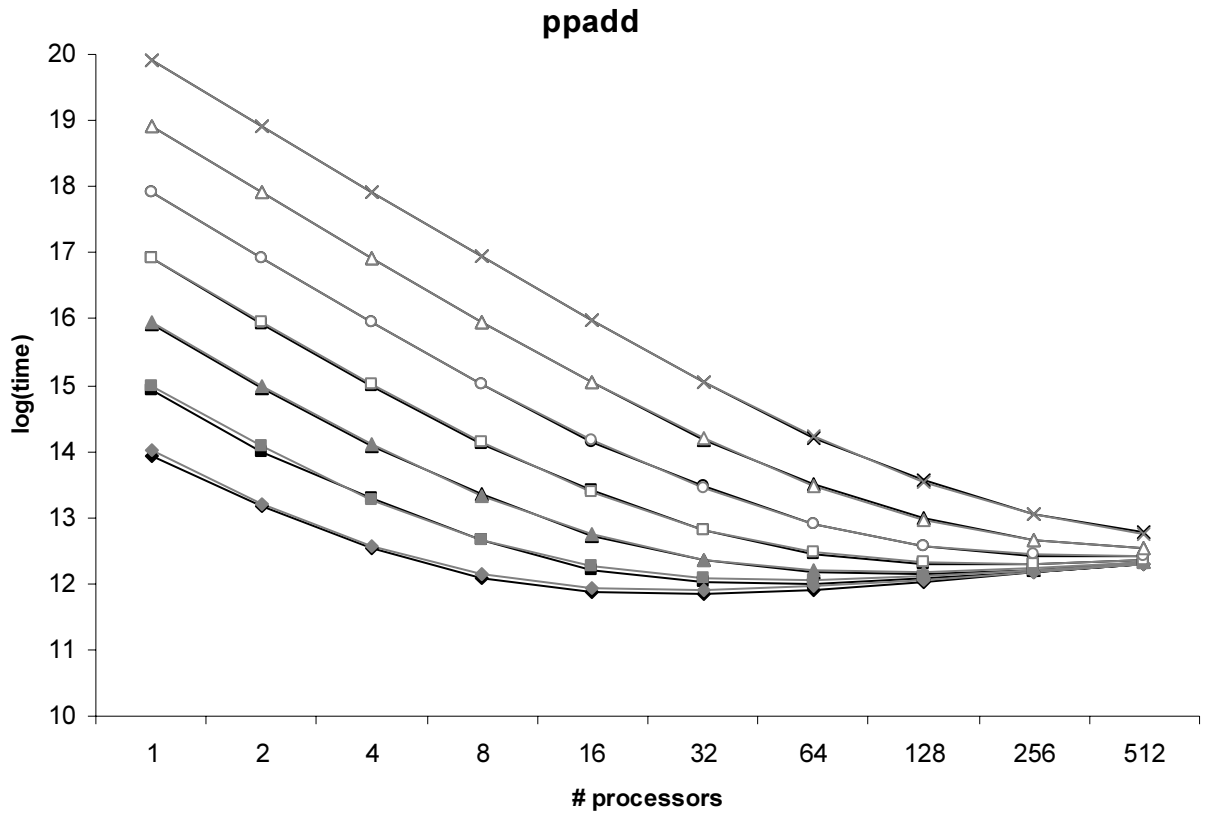
Comparison with Non-Discarding



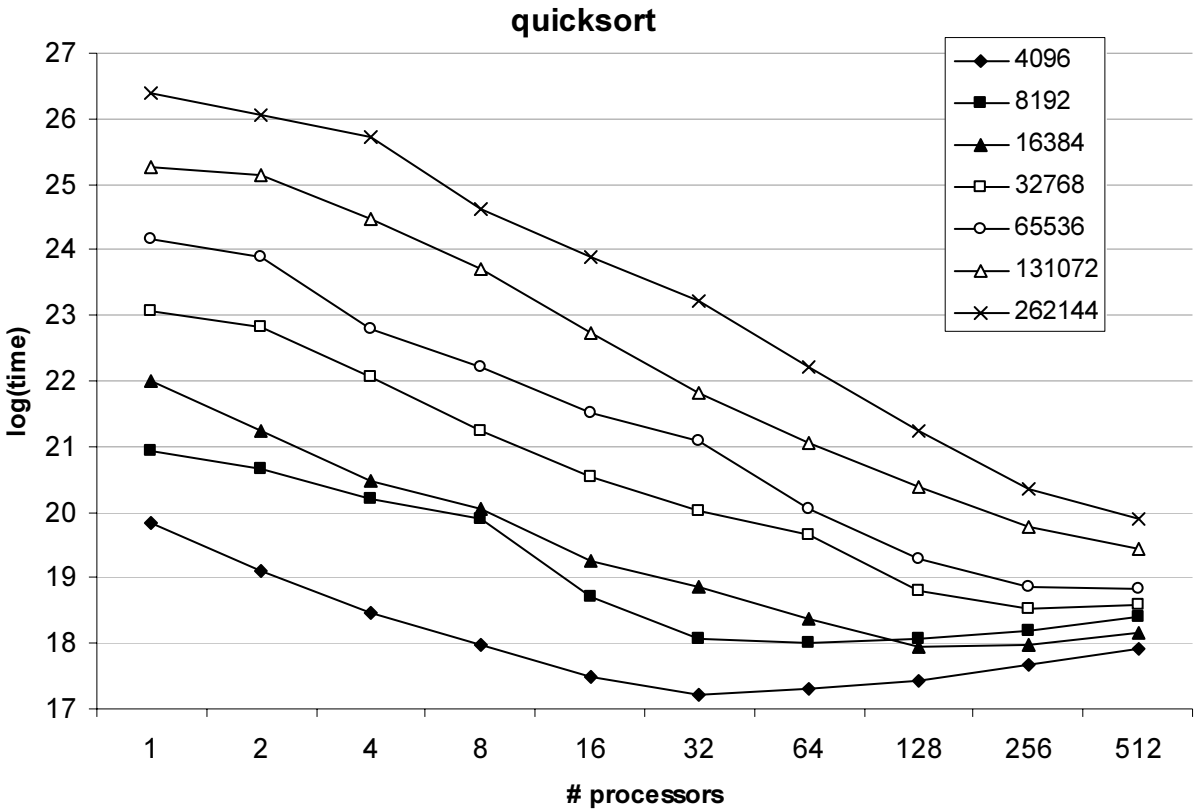
Hamal Benchmarks

- *ppadd* – Parallel-prefix addition
- *quicksort* – Parallel quicksort
- *nbody* – exact *N*-body simulation, 256 bodies
  - Processors conceptually organized in square array
  - Communication is in rows and columns only
- *wordcount* – frequency count of words in [Brown02]
  - Distributed hash table used to maintain counts

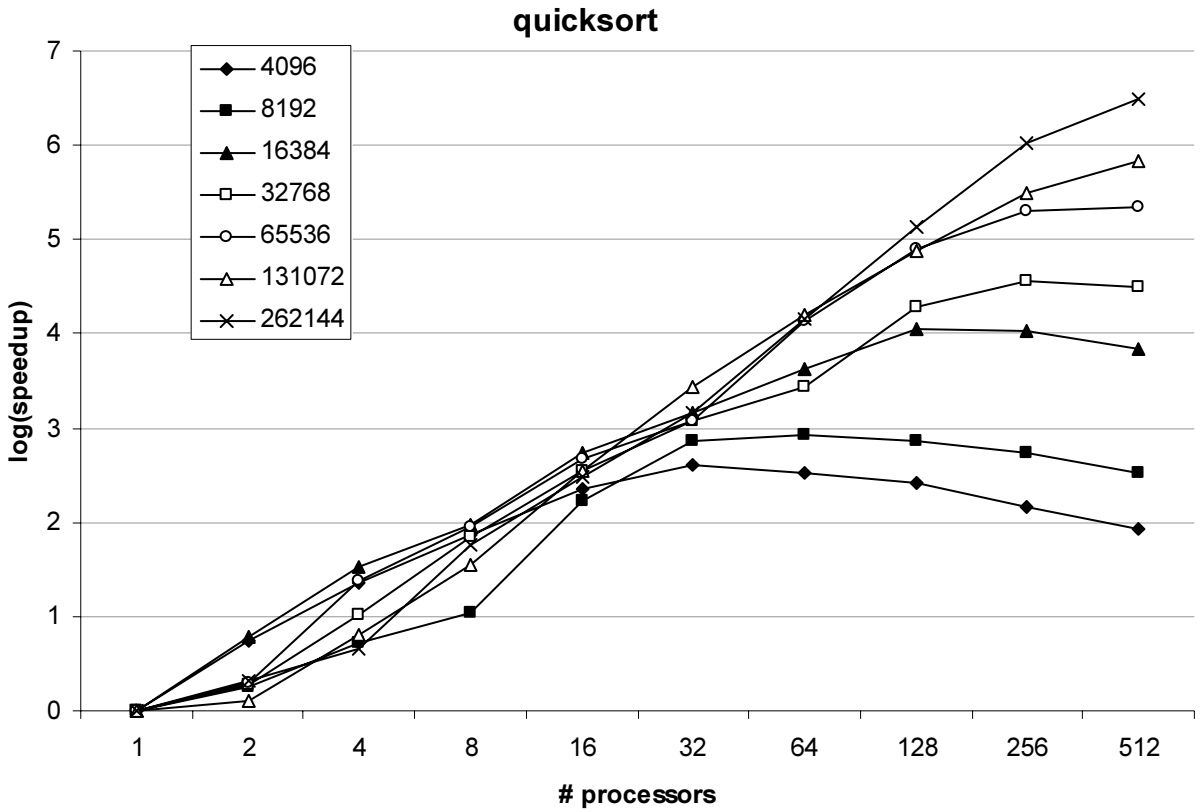
ppadd – Model vs. Simulations



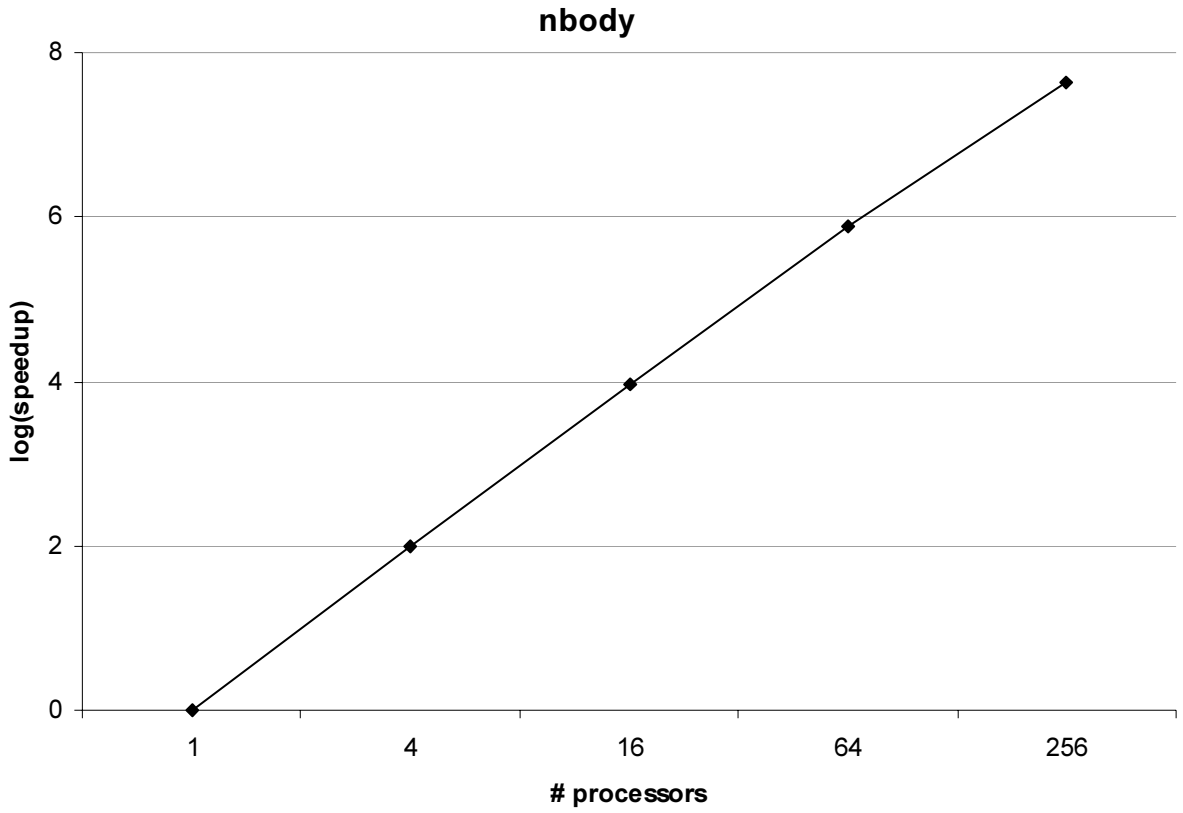
quicksort – Execution Time



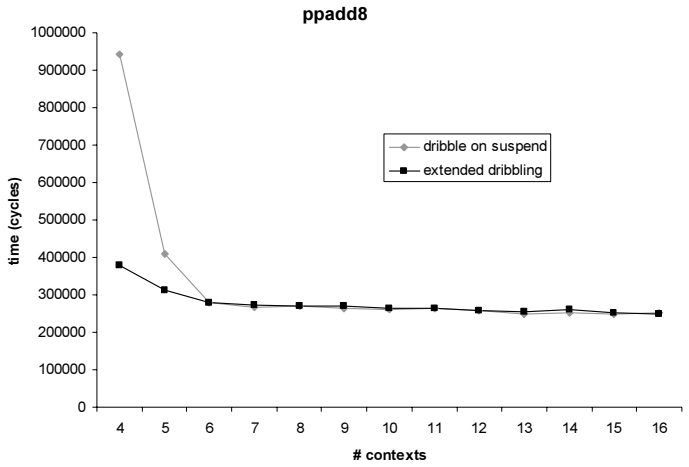
quicksort - Speedup

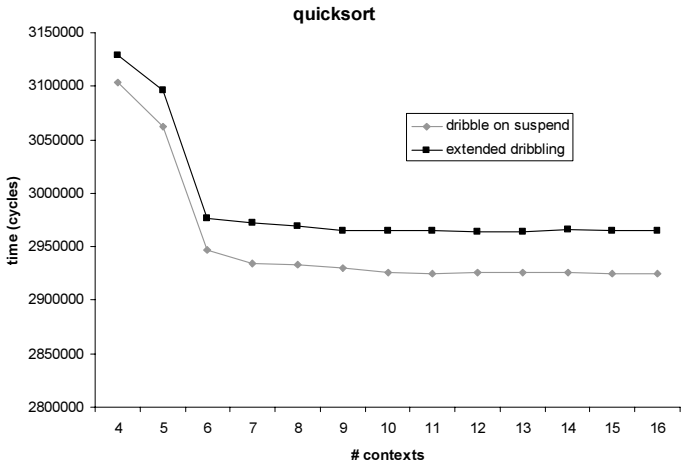


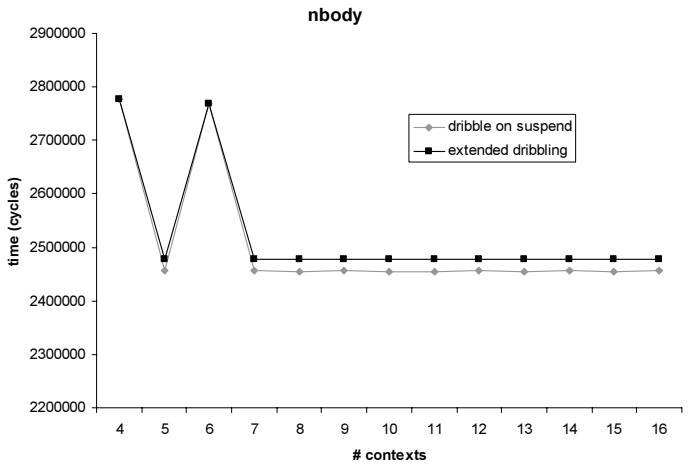
nbody – Speedup

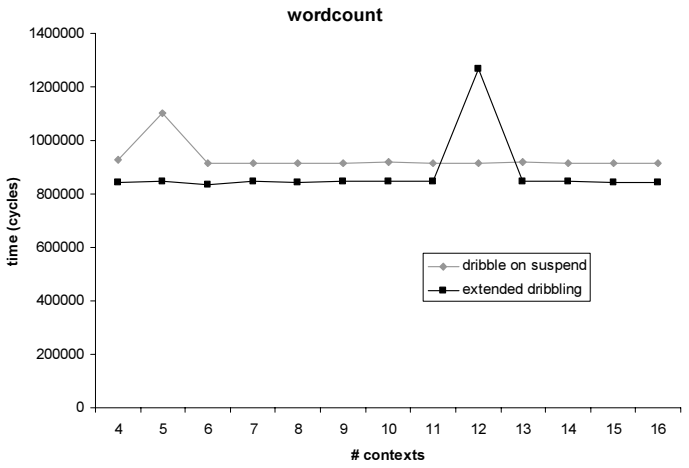


Register Dribbling









Network Benchmarks

- *add* – parallel prefix addition on 4096 nodes
- *reverse* – reverse the data of a 16K entry vector distributed across 1024 nodes
- *quicksort* – parallel quicksort of a 32K entry vector on 1024 nodes
- *nbody* – full *N*-body simulation on 256 nodes with one body per node

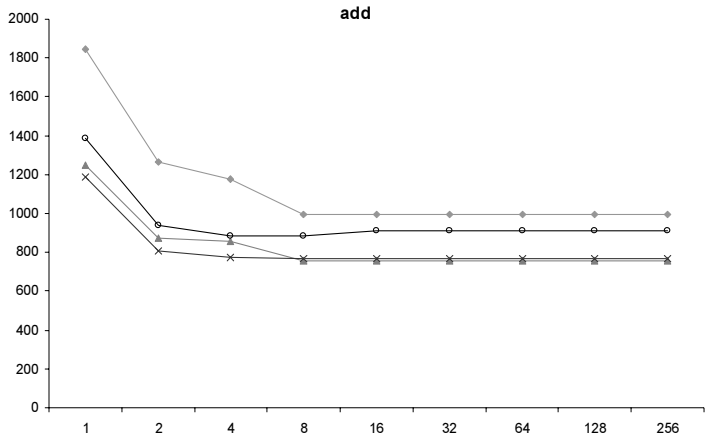
Network Topologies

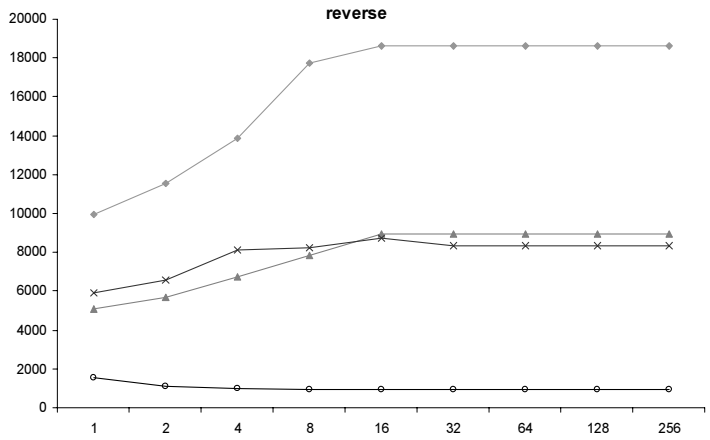
- 2D Grid
  - Dimension-ordered routing prefered
- 3D Grid
  - Dimension-ordered routing prefered
- Fat tree
  - radix-4 (down) dilation-2 (up), randomized
- Multibutterfly
  - radix-2 dilation-2, randomized

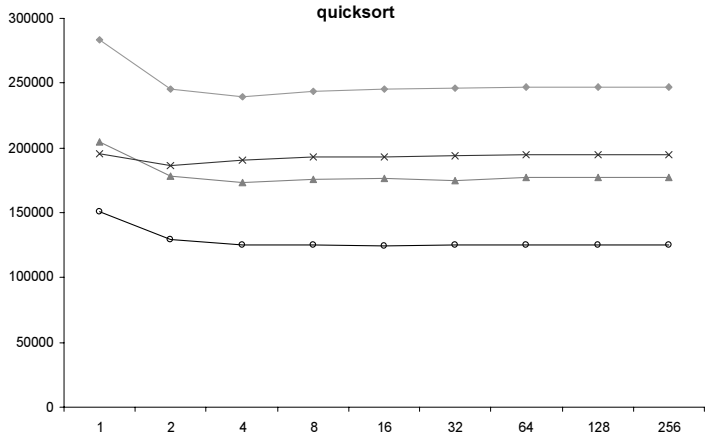
Network Retransmission

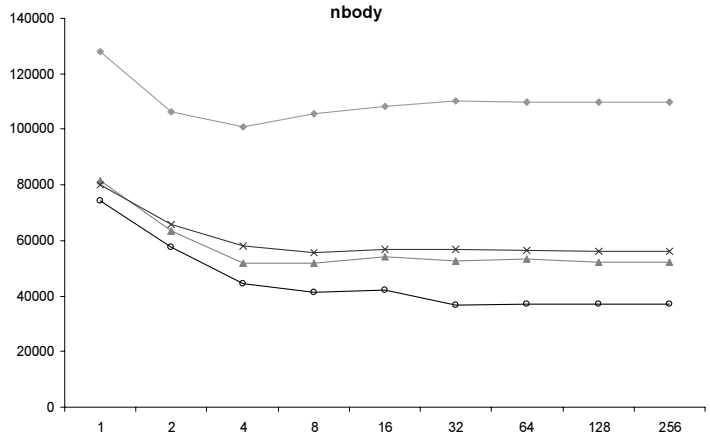
	slowdown over optimal			
	worst case		average case	
add	1.009	L5	1.003	L5
reverse	1.028	L30	1.016	L32
quicksort	1.020	Q15	1.015	Q12
nbody	1.085	L31	1.045	L31
2D grid	1.033	L32	1.026	L28
3D grid	1.039	L32	1.018	L30
fat tree	1.085	L31	1.036	L30
multibutterfly	1.041	L32	1.015	L32
overall	1.093	L30	1.028	L30

Network Send Table Size

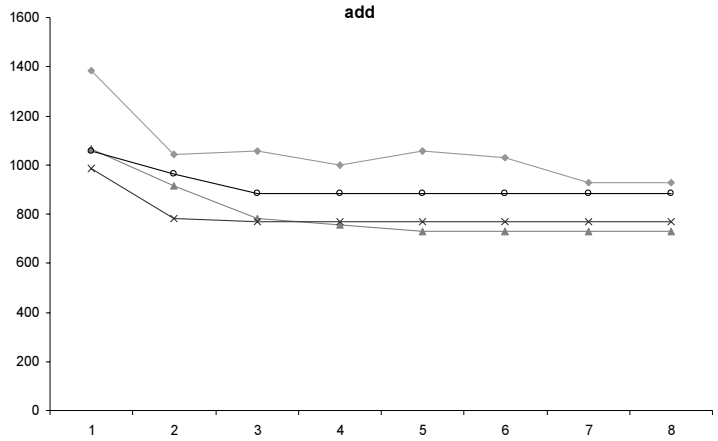


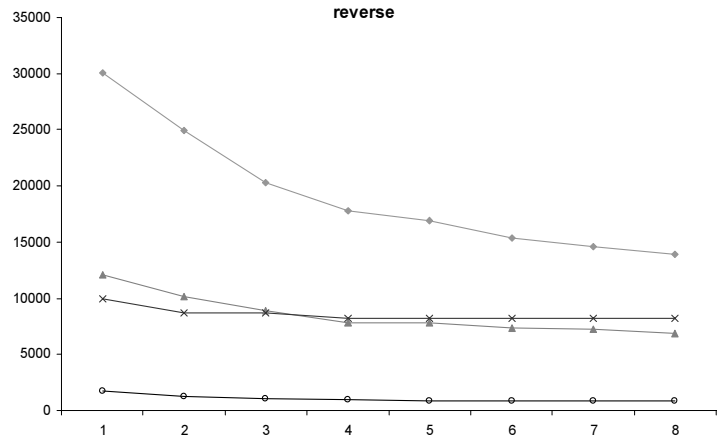


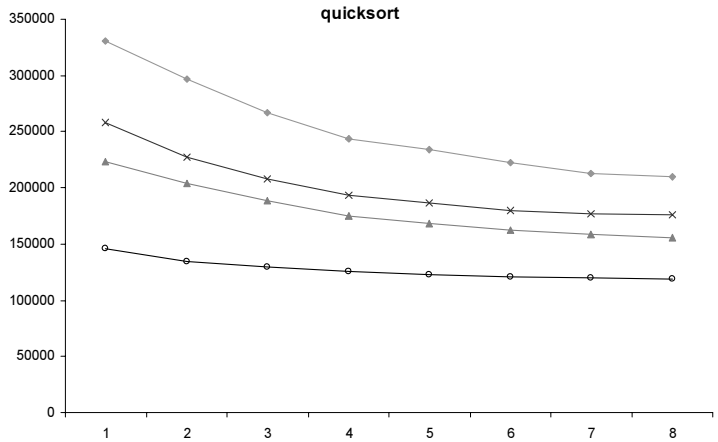


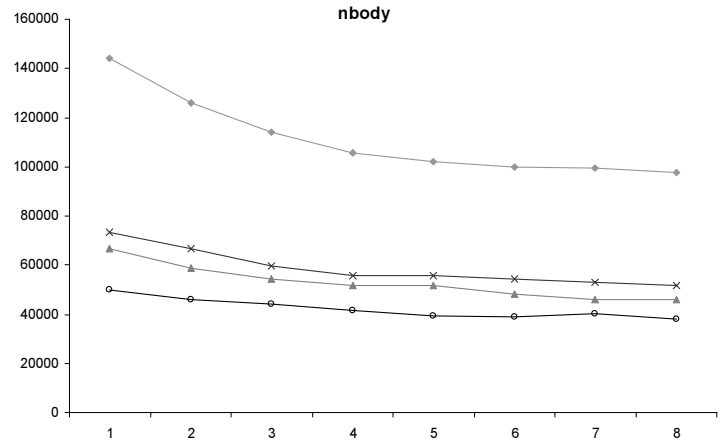


Network Buffering









Network Receive Table Size