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

virtual address

physical address

node

page

offset

virtual address

physical address

virtual page

physical page

virtual address

node

page

offset
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

doacross

Diagram showing a doacross operation with multiple processes or nodes connected in a network.
Barrier Times

![Graph showing barrier times for different numbers of processors.](image)

- **Barrier Times**
  - **X-axis:** Number of processors (1, 2, 4, 8, 16, 32, 64, 128, 256, 512)
  - **Y-axis:** Time (cycles) (0, 100, 200, 300, 400, 500, 600)
  - Data points:
    - 1 processor: 12 cycles
    - 2 processors: 58 cycles
    - 4 processors: 105 cycles
    - 8 processors: 161 cycles
    - 16 processors: 216 cycles
    - 32 processors: 273 cycles
    - 64 processors: 333 cycles
    - 128 processors: 393 cycles
    - 256 processors: 456 cycles
    - 512 processors: 523 cycles

---

December 3, 2002

The Hamal Parallel Computer
Benchmark - ppadd

The graph shows the performance of the ppadd benchmark over different numbers of processors. The performance is measured in terms of log(time) on the y-axis and the number of processors on the x-axis. The chart includes data for various processor counts: 1024, 2048, 4096, 8192, 16384, 32768, and 65536. As the number of processors increases, the log(time) decreases, indicating improved performance.
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

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

<table>
<thead>
<tr>
<th>U</th>
<th>V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>available</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>unavailable</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>trap</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>busy</td>
</tr>
</tbody>
</table>
Example – Word Locking

= thread  = word  = lock  = join capability

Diagram with symbols and arrows representing thread flow and 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

![Graph showing execution time comparison between remote and local versions.](image)
Talk Outline

✓ Overview of Hamal
✓ The Hamal memory system
✓ Threads and synchronization

- Fault-tolerant messaging protocol
## Motivation

<table>
<thead>
<tr>
<th>Discarding</th>
<th>vs.</th>
<th>Non-Discarding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td></td>
<td>Most</td>
</tr>
<tr>
<td>Examples</td>
<td>Performance</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Simplicity</td>
<td>✗</td>
</tr>
<tr>
<td></td>
<td>Reliability</td>
<td>✗</td>
</tr>
</tbody>
</table>
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

sender

MSG 7

ACK 7

CONF 7

receiver

MSG 7

MSG 7

MSG 7

CONF 7

MSG 7

MSG 7
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:**

<table>
<thead>
<tr>
<th>type</th>
<th>dest</th>
<th>source</th>
<th>message ID</th>
</tr>
</thead>
</table>

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

  **ACK:**
<table>
<thead>
<tr>
<th>type</th>
<th>dest</th>
<th>source</th>
<th>message ID</th>
<th>ID2</th>
</tr>
</thead>
</table>

  **CONF:**
<table>
<thead>
<tr>
<th>type</th>
<th>dest</th>
<th>ID2</th>
</tr>
</thead>
</table>
Failure of Short IDs

sender

MSG 7

ACK 7

receiver

MSG 7

CONF 7

ACK 7

MSG 7

CONF 7

MSG 7

ACK 7

MSG 7

CONF 7

MSG 7
Secondary IDs

sender

MSG 5

MSG 5

ACK 5,2

MSG 5,2

MSG 9

CONF 2

MSG 9

MSG 9

CONF 2

ACK 9,2

MSG 9,2

MSG 9

ACK 5,2

MSG 5,2

MSG 5

CONF 2

MSG 9

ACK 9,2

MSG 9

receiver
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

<table>
<thead>
<tr>
<th></th>
<th>Non-Discarding</th>
<th>Discarding + fault-tolerant messaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reverse</td>
<td></td>
<td></td>
</tr>
<tr>
<td>quicksort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nbody</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The graphs show the performance comparison between Non-Discarding and Discarding + fault-tolerant messaging for different grid topologies and applications.
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

![Graph showing quicksort execution time vs. number of processors. The x-axis represents the number of processors ranging from 1 to 512, and the y-axis represents the logarithm of the time. The graph displays multiple lines each representing a different number of elements: 4096, 8192, 16384, 32768, 65536, 131072, 262144. Each line shows a decrease in time as the number of processors increases.]
quicksort - Speedup

![Graph showing the speedup of quicksort with increasing number of processors. The x-axis represents the number of processors ranging from 1 to 512, and the y-axis represents the log of speedup. Different line styles correspond to different data points, indicating varying performance across different processor counts.](quicksort_speedup.png)
nbody – Speedup

![Graph showing the speedup of nbody with increasing number of processors. The graph plots log(speedup) against the number of processors. There is a linear relationship observed.](image)
Register Dribbling

- **ppadd8**
  - Time (cycles) vs. # contexts
  - Dribble on suspend: ▲
  - Extended dribbling: ■

- **quicksort**
  - Time (cycles) vs. # contexts
  - Dribble on suspend: ▲
  - Extended dribbling: ■

- **nbody**
  - Time (cycles) vs. # contexts
  - Dribble on suspend: ▲
  - Extended dribbling: ■

- **wordcount**
  - Time (cycles) vs. # contexts
  - Dribble on suspend: ▲
  - Extended 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 preferred

- **3D Grid**
  - Dimension-ordered routing preferred

- **Fat tree**
  - radix-4 (down) dilation-2 (up), randomized

- **Multibutterfly**
  - radix-2 dilation-2, randomized
## Network Retransmission

<table>
<thead>
<tr>
<th></th>
<th>slowdown over optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>worst case</td>
</tr>
<tr>
<td><strong>add</strong></td>
<td>1.009</td>
</tr>
<tr>
<td><strong>reverse</strong></td>
<td>1.028</td>
</tr>
<tr>
<td><strong>quicksort</strong></td>
<td>1.020</td>
</tr>
<tr>
<td><strong>nbody</strong></td>
<td>1.085</td>
</tr>
<tr>
<td><strong>2D grid</strong></td>
<td>1.033</td>
</tr>
<tr>
<td><strong>3D grid</strong></td>
<td>1.039</td>
</tr>
<tr>
<td><strong>fat tree</strong></td>
<td>1.085</td>
</tr>
<tr>
<td><strong>multibutterfly</strong></td>
<td>1.041</td>
</tr>
<tr>
<td><strong>overall</strong></td>
<td>1.093</td>
</tr>
</tbody>
</table>
Network Send Table Size

- **add**
  - Network Send Table Size
  - Network Send Table Size

- **reverse**
  - Network Send Table Size
  - Network Send Table Size

- **quicksort**
  - Network Send Table Size
  - Network Send Table Size

- **nbody**
  - Network Send Table Size
  - Network Send Table Size
Network Buffering

![Graphs for add, reverse, quicksort, and nbody operations showing performance metrics over different data points.](image-url)
Network Receive Table Size