Evaluation and Modeling of Program Execution Models

Stéphane Zuckerman

Computer Architecture & Parallel Systems Laboratory
Department of Electrical and Computer Engineering
University of Delaware

October 19th, 2011
Outline

Looking at a PXM Abstract Machine again

Where to Implement a PXM?
  - Full Hardware Implementation
  - Full Software Implementation
  - Hardware-Software Co-Design
  - Timeline

Evaluating PXMs’ Efficiency [4, 9]
  - Analytical Models
  - Micro-Benchmarking
  - Application Benchmarking
  - Evaluating Extensions to a given couple PXM-Abstract Machine
Relationship Between PXMs and Actual Computer Systems

Execution Model

Programming Environment Platforms

Execution Model API

Abstract Machine

Programming Models

Users

Execution Model and Abstract Machines
Relationship Between PXMs and Actual Computer Systems

Looking at a PXM Abstract Machine again

Where to Implement a PXM?
- Full Hardware Implementation
- Full Software Implementation
- Hardware-Software Co-Design
- Timeline

Evaluating PXMs’ Efficiency
- [4, 9]
  - Analytical Models
  - Micro-Benchmarking
  - Application Benchmarking
  - Evaluating Extensions to a given couple PXM-Abstract Machine
Outline

Looking at a PXM Abstract Machine again

Where to Implement a PXM?
- Full Hardware Implementation
- Full Software Implementation
- Hardware-Software Co-Design
- Timeline

Evaluating PXMs’ Efficiency [4, 9]
- Analytical Models
- Micro-Benchmarking
- Application Benchmarking
- Evaluating Extensions to a given couple PXM-Abstract Machine
Full Hardware Implementation

Pros

- Would seem like the most efficient method: No additional software layer between the programmer and the hardware
- HW and abstract machines are a 1:1 match

Cons

- Any mistake in hardware is *costly*
  - Bug in the implementation
  - Conceptual mistake in the design
- Needs a “perfect” design beforehand
- Not always possible financially
- Makes the implementation of other PXMs potentially more difficult (not necessarily a weakness)
Full Software Implementation

Pros

- Very flexible: any hardware architecture can be targeted
- Any oversight in the design of the PXM can be fixed relatively easily

Cons

- Some operations can be very slow if not implemented in hardware
- Can force the high-level programmers to know more about "gory details" than they should in order to make programs run efficiently
Trade-offs must be found (e.g., atomic instructions to help build fast lock operations)

Needs ways to model, measure and evaluate how well a given PXM and its associated abstract machine perform in order to decide what to implement in SW or HW.
Timeline

Where to Implement a PXM?

Timeline

Evaluating PXMs’ Efficiency

[4, 9]

Analytical Models
Micro-Benchmarking
Application Benchmarking
Evaluating Extensions to a given couple
PXMs-Abstract Machine

10/19/2011 9 / 52
Outline

Looking at a PXM Abstract Machine again

Where to Implement a PXM?
  - Full Hardware Implementation
  - Full Software Implementation
  - Hardware-Software Co-Design
  - Timeline

Evaluating PXMs’ Efficiency [4, 9]
  - Analytical Models
  - Micro-Benchmarking
  - Application Benchmarking
  - Evaluating Extensions to a given couple PXM-Abstract Machine

S.Zuckerman (University of Delaware)
PXM evaluation & modeling
10/19/2011 10 / 52
Analytical Models

Description & Purpose

- Based on solid mathematical (often probabilistic/statistical) methods
- For specific features to evaluate
- Provide very useful trends for a given mechanism (when done right)
- Can give very accurate information on the behavior of a system (e.g., queueing networks)
- Shows its limits when trying to apply to a full system which implements the whole PXM (too many parameters)
Micro-Benchmarks

- Made to evaluate the overhead induced by individual constructs of the PXM
- They only *verify* a given implementation is efficient, they do not *validate* the PXM does what it is intended to do
- Helps to predict the *minimal* overhead to expect when using the PXM
Purpose of Application Benchmarking

- Must be representative of the kind of workload the PXM should process
- Helps determine how close (or far) the PXM is from fulfilling its goals – and how efficiently: programmability-wise, speed-wise, etc.
What to Measure

For parallel workloads

- Sequential execution ($SE_{init}$): provide a baseline
- Sequential execution programmed with the PXM ($SE_{PXM}$): measure the *global* overhead of the PXM
- Parallel execution programmed with the PXM ($PE_{PXM}$)

Time Criterion Example

- $SE_{init} / SE_{PXM}$ gives the global overhead of the given PXM
- $SE_{init} / PE_{PXM}$ gives the *absolute* speedup of the PXM
- $SE_{PXM} / PE_{PXM}$ gives the *relative* speedup of the PXM
Evaluating Extensions to a given couple PXM-Abstract Machine

Motivation

- Current implementation may incur too much overhead for certain constructs
- Hardware is not necessarily available to test new ideas

Use of simulation

- Function-accurate
- Cycle-accurate
- Gate-accurate
Evaluating Extensions to a given couple PXM-Abstract Machine

Motivation

- Current implementation may incur too much overhead for certain constructs
- Hardware is not necessarily available to test new ideas

Use of simulation

- Function-accurate
- Cycle-accurate
- Gate-accurate
Case studies: OpenMP and EARTH

OpenMP
- Share-memory programming model
- One of the most popular (and available) programming models out there

EARTH
- Already seen before
- Hybrid Von Neumann – data flow model of computation
- Evaluated in multiple ways
Outline

The OpenMP Execution Model
Evaluating OpenMP’s efficiency
Application Benchmarking with OpenMP
Extending OpenMP

Evaluating EARTH
Analytical Models for EARTH
Evaluating EARTH on Off-the-Shelf Computers
Other Ports of EARTH [15]
Extending Hardware to be EARTH-compliant
Overview

The OpenMP Programming Model [5]

- No specific abstract machine model (relies on Von Neumann’s model for threads/processors)
- A language extension to Fortran, C, C++
- A library
- A runtime system

Originally, it was made to express data-parallel and SPMD programs easily.
Threading Model: Fork-Join

```c
#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < M; ++i)
        for (int j = 0; j < N; ++j)
            for (int k = 0; k < K; ++k)
                C[i*N+j] = \beta \cdot C[i*N+j] + \alpha \cdot A[i*K+k] \cdot B[k*N+j];
}
```
Memory Model [7] & Synchronization API

Directory/Clause | Effect
--- | ---
nowait | Removes the implicit barrier of a given directive/clause
flush(v1,v2,...) | Forces the variables $v_i$ to be written to (read from) memory (commits these variables from the temporary view to the shared memory).
critical [(name)] | Declares a given section of code is a critical section. Only one thread can go in at a time.

Library Call | Effect
--- | ---
omp_set_lock (omp_lock_t* lock) | Tries to acquire lock $lock$
omp_unset_lock (omp_lock_t* lock) | Releases a lock $lock$

Table: Example of directives and library calls for synchronization in OpenMP

Reminder: this is not the complete description of the OpenMP model!
Microbenchmarking: Using EPCC [3]

Description

- EPCC microbenchmarks (Edinburgh Parallel Computing Center) evaluate various overheads:
  - Scheduling policies (static, dynamic, guided)
  - Synchronization directives (barrier, single/master, atomic/critical)
  - Privatization directives (private, firstprivate, lastprivate, copyprivate, threadprivate)
- Provides a way to compare different implementations of OpenMP
  - same hardware platform (eg: gcc vs icc)
  - same compiler (eg Itanium2 vs Core 2 Quad)
Experimental Testbed

Itanium2

- EPIC architecture (VLIW + superscalar)
- Mostly in-order (except for memory operations)
- All caches are private (16KB/256KB/12MB)
- Heat sink (Intel could never go beyond 1.6 GHz)
- Montecito and Montvale differ only w.r.t. the memory bus frequency (533MHz vs 667MHz).
- 2 types of nodes: UMA (Montecito) and NUMA (Montecito, Montvale)

Xeon Woodcrest

- Core 2 family (x86, out-of-order, superscalar, etc.)
- Private L1 cache: 32 KB
- Last level of cache (L2, 4MB) is shared between the 2 cores

Software

- OS: Linux (kernel 2.6.18)
- Compiler: ICC v10.0
Example of Results with EPCC

Figure: IA64

Figure: x86

arraybench results
Example of Results with EPCC

Figure: IA64

syncbench results

Figure: x86
Example of Results with EPCC

Figure: IA64

atomic results

Figure: x86
Example of Results with EPCC

**Figure: IA64**

**Figure: x86**

schedbench results
## Application Benchmarking with OpenMP

### Table: NASA Advanced Supercomputing (NAS) Parallel Benchmarks [2, 10]

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>Simulated CFD: 3D Navier-Stoke equations. Alternating Direction Implicit (ADI) used to solve the finite difference solution to the problem.</td>
</tr>
<tr>
<td>SP</td>
<td>Simulated CFD: uses Beam-Warming approximate factorization to solve the finite difference problem.</td>
</tr>
<tr>
<td>LU</td>
<td>Simulated CFD: uses symmetric successive over-relaxation (SSOR) to solve a 3D Navier-Stoke equation system. Uses LU matrix decomposition kernels.</td>
</tr>
<tr>
<td>FT</td>
<td>3D Fast Fourier Transform (FFT). Based on spectral methods.</td>
</tr>
<tr>
<td>MG</td>
<td>3D scalar Poisson equation. solved with a V-cycle MultiGrid method.</td>
</tr>
<tr>
<td>CG</td>
<td>Conjugate Gradient used to compute the smallest eigenvalue of a large, sparse, unstructured matrix.</td>
</tr>
<tr>
<td>EP</td>
<td>Embarrassingly Parallel benchmark. Goal: provide reference point for all other benchmarks.</td>
</tr>
</tbody>
</table>

### Table: SPEComp benchmarks [1]

<table>
<thead>
<tr>
<th>Name</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>ammp</td>
<td>Chemistry/biology</td>
</tr>
<tr>
<td>applu</td>
<td>Fluid dynamics/physics</td>
</tr>
<tr>
<td>apsi</td>
<td>Air pollution</td>
</tr>
<tr>
<td>art</td>
<td>Image recognition/neural networks</td>
</tr>
<tr>
<td>facerec</td>
<td>Face recognition</td>
</tr>
<tr>
<td>fma3d</td>
<td>Crash simulation</td>
</tr>
<tr>
<td>gafort</td>
<td>Genetic algorithm</td>
</tr>
<tr>
<td>galgel</td>
<td>Fluid dynamics</td>
</tr>
<tr>
<td>equake</td>
<td>Earthquake modeling</td>
</tr>
<tr>
<td>mgrid</td>
<td>Multigrid solver</td>
</tr>
<tr>
<td>swim</td>
<td>Shallow water modeling</td>
</tr>
<tr>
<td>wupwise</td>
<td>Quantum chromodynamics (QCD)</td>
</tr>
</tbody>
</table>
Extending OpenMP

- Nested parallelism (OpenMP 2.0-2.5)
  - Not implemented in all OpenMP runtime systems yet (it is optional in the standard)
  - Can help handle “static” outer scheduling but “dynamic” inner scheduling
- Going beyond data/loop parallelism: tasks [6] (OpenMP 3.0)
  - Can “flatten” recursive calls
  - Created to handle pointer-chasing
  - For now, performance is rather poor [12]
- Loop coalescing directive (OpenMP 3.0)
- See [http://www.openmp.org](http://www.openmp.org)
- Mostly an “evolution” rather than a “revolution”
Outline

The OpenMP Execution Model
   Evaluating OpenMP’s efficiency
   Application Benchmarking with OpenMP
   Extending OpenMP

Evaluating EARTH
   Analytical Models for EARTH
   Evaluating EARTH on Off-the-Shelf Computers
   Other Ports of EARTH [15]
   Extending Hardware to be EARTH-compliant
Analytical Models Applied to EARTH (and HTMT)

- Closed Queuing Network theory [11]: models EUs, SUs, output messages, input messages, under certain constraints

- Evaluation of the benefits of percolation [8]. The model predicts potential speedups going from 2 to 11 depending on memory behaviors of the programs, and how high memory latencies are.
EARTH-MANNA [13]

The MANNA supercomputer

- Made out of Intel i860 XP processors
  - RISC
  - clocked at 50MHz
  - 16KB L1 cache
- Each node embeds
  - 32MB
  - 2 processors
  - Cache coherence using MESI
  - Custom-designed link chip (memory-interconnect interface)
  - connected to other nodes through a $16 \times 16$ crossbar

Figure: A MANNA node.
Microbenchmark Example: ping-pong

Parameter | Dual-processor | Single-processor |
--- | --- | --- |
Latency (ns) | 4091 | 2450 |
Latency (cycles) | 204.5 | 122.5 |
Bandwidth (MB/s) | 42.0 | 28.8 |
Bandwidth (% of peak) | 83.9 | 57.5 |

Table: Latency and Bandwidth on EARTH-MANNA
## Microbenchmarks: Operation Latencies

<table>
<thead>
<tr>
<th>Operation</th>
<th>Dual-processor nodes</th>
<th></th>
<th>Single-processor nodes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Sequential</td>
<td>Pipelined</td>
<td>Sequential</td>
</tr>
<tr>
<td>(r)sync</td>
<td>2327</td>
<td>3982</td>
<td>841</td>
<td>994</td>
</tr>
<tr>
<td>(r)spawn</td>
<td>2252</td>
<td>4266</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>get_sync</td>
<td>2824</td>
<td>6968</td>
<td>1137</td>
<td>1880</td>
</tr>
<tr>
<td>data_(r)sync</td>
<td>2767</td>
<td>6667</td>
<td>1060</td>
<td>1814</td>
</tr>
<tr>
<td>invoke (1 arg)</td>
<td>5011</td>
<td>9011</td>
<td>3188</td>
<td>2794</td>
</tr>
<tr>
<td>invoke (5 args)</td>
<td>6217</td>
<td>10240</td>
<td>3879</td>
<td>2984</td>
</tr>
<tr>
<td>invoke (9 args)</td>
<td>6826</td>
<td>10727</td>
<td>4260</td>
<td>3504</td>
</tr>
<tr>
<td>invoke (18 args)</td>
<td>8192</td>
<td>12552</td>
<td>5529</td>
<td>4456</td>
</tr>
</tbody>
</table>

**Table: EARTH Operation Latencies (nsec.) on EARTH-MANNA**
# Microbenchmarks: EU Costs of EARTH Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Dual-processor nodes</th>
<th>Single-processor nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Local</td>
<td>Remote</td>
</tr>
<tr>
<td>(r)sync</td>
<td>504</td>
<td>504</td>
</tr>
<tr>
<td>(r)spawn</td>
<td>721</td>
<td>580</td>
</tr>
<tr>
<td>end_fiber</td>
<td>530</td>
<td>N/A</td>
</tr>
<tr>
<td>incr_(r)sync</td>
<td>561</td>
<td>554</td>
</tr>
<tr>
<td>data_(r)sync</td>
<td>580</td>
<td>606</td>
</tr>
<tr>
<td>get_sync</td>
<td>580</td>
<td>620</td>
</tr>
<tr>
<td>invoke (1 arg)</td>
<td>760</td>
<td>620</td>
</tr>
<tr>
<td>end_procedure (1 arg)</td>
<td>794</td>
<td>N/A</td>
</tr>
<tr>
<td>invoke (5 args)</td>
<td>1039</td>
<td>907</td>
</tr>
<tr>
<td>end_procedure (5 args)</td>
<td>1203</td>
<td>N/A</td>
</tr>
<tr>
<td>invoke (9 args)</td>
<td>1223</td>
<td>1210</td>
</tr>
<tr>
<td>end_procedure (9 args)</td>
<td>1372</td>
<td>N/A</td>
</tr>
<tr>
<td>invoke (18 args)</td>
<td>1766</td>
<td>1512</td>
</tr>
<tr>
<td>end_procedure (18 args)</td>
<td>1728</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Table:** EARTH-MANNA-D: Cost of forming a request message and writing it to the EQ in memory; for EARTH-MANNA-S: Cost of stopping and performing the entire operation (if local) or forming a request message and writing it to the link chip (if remote)
# Application Benchmarking: Sequential Timings

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Input</th>
<th>$T_{seq}$ (sec.)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>$2^{16}$</td>
<td>0.866</td>
<td>Regular; frequent data moves</td>
</tr>
<tr>
<td>Fibonacci</td>
<td>30</td>
<td>0.969</td>
<td>Recursive; high overheads</td>
</tr>
<tr>
<td>Matrix multiply</td>
<td>$512 \times 512$</td>
<td>36.6</td>
<td>Regular, data-parallel</td>
</tr>
<tr>
<td>N-Queens-P</td>
<td>12 queens</td>
<td>17.2</td>
<td>Fully para. recursive enumeration</td>
</tr>
<tr>
<td>N-Queens-T</td>
<td>12 queens</td>
<td></td>
<td>Partially sequentialized</td>
</tr>
<tr>
<td>Paraffins</td>
<td>$N = 23$</td>
<td>3.69</td>
<td>Recursive enumeration</td>
</tr>
<tr>
<td>Povray</td>
<td>shapes $(256)^2$</td>
<td>69.4</td>
<td>Task-parallel</td>
</tr>
<tr>
<td>Protein folding</td>
<td>$3 \times 3 \times 3$</td>
<td>7.43</td>
<td>Recursive search</td>
</tr>
<tr>
<td>SLT-2D</td>
<td>$80 \times 80$</td>
<td>2.60</td>
<td>Regular, data-parallel</td>
</tr>
<tr>
<td>Tomcatv</td>
<td>$N = 257$</td>
<td>48.6</td>
<td>Regular, data-parallel, barrier</td>
</tr>
<tr>
<td>TSP</td>
<td>10 cities</td>
<td>38.2</td>
<td>Recursive search</td>
</tr>
</tbody>
</table>

**Table**: Benchmarks and Sequential Performance
Metrics to Measure EARTH-MANNA’s Performance

The USE factor

\[ USE = \frac{T_{seq}}{T_1}, \]

- \( T_{seq} \): best “pure” sequential execution time
- \( T_1 \): execution time using EARTH (Threaded-C program) with a single thread

Parallel Performance Metrics

- Relative speedup on \( k \) nodes: \( R_k = \frac{T_1}{T_k} \)
- Absolute speedup on \( k \) nodes: \( A_k = \frac{T_{seq}}{T_k} \)
- Relationship between \( R_k \) and \( A_k \): \( A_k = USE \times R_k \)
### Application Benchmarking: Uni-Node Support Efficiencies aka USE Factor

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>USE factor (%)</th>
<th>Dual-processor</th>
<th>Single-processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>59.8</td>
<td>75.6</td>
<td></td>
</tr>
<tr>
<td>Fibonacci</td>
<td>7.55</td>
<td>13.9</td>
<td></td>
</tr>
<tr>
<td>Matrix multiply</td>
<td>99.9</td>
<td>100.3</td>
<td></td>
</tr>
<tr>
<td>N-Queens-P</td>
<td>52.5</td>
<td>67.0</td>
<td></td>
</tr>
<tr>
<td>N-Queens-T</td>
<td>98.8</td>
<td>99.3</td>
<td></td>
</tr>
<tr>
<td>Paraffins</td>
<td>91.4</td>
<td>99.4</td>
<td></td>
</tr>
<tr>
<td>Povray</td>
<td>94.0</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>Protein folding</td>
<td>95.0</td>
<td>98.8</td>
<td></td>
</tr>
<tr>
<td>SLT-2D</td>
<td>88.5</td>
<td>99.9</td>
<td></td>
</tr>
<tr>
<td>Tomcatv</td>
<td>95.0</td>
<td>100.0</td>
<td></td>
</tr>
<tr>
<td>TSP</td>
<td>98.9</td>
<td>99.6</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Uni-Node Support Efficiencies on EARTH-MANNA
Application Benchmarking: Relative Speedups

Figure: Single-processor

Figure: Dual-processor
Application Benchmarking: Absolute Speedups

Figure: Single-processor

Figure: Dual-processor
Other Ports of EARTH

EARTH on IBM SP2

- Implied changes to Threaded-C (32 bit address space not enough to address more than 4GB)
- Compilation chain changed due to different ISA

EARTH-Beowulf

- Network-of-Workstations
- Fast Ethernet (100Base-T)
- 60-node machine running Povray (presented at CalTech in 1998)
- Inter-node communications pass through TCP/IP

Clusters of SMP Workstations

- 4-way UltraSPARC-II machines
- Shared memory (local crossbar)
- Myrinet network interconnect
- Reuses EARTH-Beowulf implementation
- Handles multiple EUs
Other Ports of EARTH

EARTH on IBM SP2

- Implied changes to Threaded-C (32 bit address space not enough to address more than 4GB)
- Compilation chain changed due to different ISA

EARTH-Beowulf

- Network-of-Workstations
- Fast Ethernet (100Base-T)
- 60-node machine running Povray (presented at CalTech in 1998)
- Inter-node communications pass through TCP/IP

Clusters of SMP Workstations

- 4-way UltraSPARC-II machines
- Shared memory (local crossbar)
- Myrinet network interconnect
- Reuses EARTH-Beowulf implementation
- Handles multiple EUs
Other Ports of EARTH

EARTH on IBM SP2

- Implied changes to Threaded-C (32 bit address space not enough to address more than 4GB)
- Compilation chain changed due to different ISA

EARTH-Beowulf

- Network-of-Workstations
- Fast Ethernet (100Base-T)
- 60-node machine running Povray (presented at CalTech in 1998)
- Inter-node communications pass through TCP/IP

Clusters of SMP Workstations

- 4-way UltraSPARC-II machines
- Shared memory (local crossbar)
- Myrinet network interconnect
- Reuses EARTH-Beowulf implementation
- Handles multiple EUs
Extending Hardware to be EARTH-compliant

Why Extend EARTH?

- EARTH was designed to run on off-the-shelf multiprocessor computers
- What if a specialized computer was built for EARTH?
- Use of SEMi [14]: Simulator of EARTH-MANNA on i860 (single-threaded, cycle-accurate to some degree)
- Speed ratio: $\approx 300 - 500$ times slower than reality (which is not bad!)

Additional Hardware Features

- Extension of the machine from 20 to 120 nodes
- Modification of the i860:
  - Models changes to the network topology ($n \times n$ network of routers)
  - Parameterized caches and memory delays
  - Added scoreboard logic (instead of locking the whole functional unit)
  - Non-blocking on-chip L1 cache
  - Added an L2 cache
  - Added in-order, multiple instruction issue (instead of the limited VLIW capabilities of the i860)
Extending Hardware to be EARTH-compliant

Why Extend EARTH?

- EARTH was designed to run on off-the-shelf multiprocessor computers
- What if a specialized computer was built for EARTH?
- Use of SEMi [14]: Simulator of EARTH-MANNA on i860 (single-threaded, cycle-accurate to some degree)
- Speed ratio: $\approx 300 - 500$ times slower than reality (which is not bad!)

Additional Hardware Features

- Extension of the machine from 20 to 120 nodes
- Modification of the i860:
  - Models changes to the network topology ($n \times n$ network of routers)
  - Parameterized caches and memory delays
  - Added scoreboard logic (instead of locking the whole functional unit)
  - Non-blocking on-chip L1 cache
  - Added an L2 cache
  - Added in-order, multiple instruction issue (instead of the limited VLIW capabilities of the i860)
## Results after simulation: USE Factor

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Input</th>
<th>$T_{seq}$ (sec)</th>
<th>USE factor (%)</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dual-processor</td>
<td>Single-processor</td>
<td></td>
</tr>
<tr>
<td>Fibonacci</td>
<td>15</td>
<td>0.000831</td>
<td>8.6</td>
<td>15.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.00801</td>
<td>7.7</td>
<td>14.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>0.0875</td>
<td>7.6</td>
<td>13.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.969</td>
<td>7.6</td>
<td>13.9</td>
<td></td>
</tr>
<tr>
<td>N-Queens-P</td>
<td>8</td>
<td>0.0223</td>
<td>39.9</td>
<td>51.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.541</td>
<td>46.8</td>
<td>56.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>17.3</td>
<td>53.9</td>
<td>65.6</td>
<td></td>
</tr>
<tr>
<td>N-Queens-T</td>
<td>8</td>
<td>0.0223</td>
<td>68.5</td>
<td>78.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.541</td>
<td>93.1</td>
<td>95.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>17.3</td>
<td>99.1</td>
<td>99.3</td>
<td></td>
</tr>
<tr>
<td>Paraffins</td>
<td>18</td>
<td>0.0394</td>
<td>82.1</td>
<td>97.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.228</td>
<td>85.4</td>
<td>101.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>3.69</td>
<td>84.7</td>
<td>100.6</td>
<td></td>
</tr>
<tr>
<td>Tomcatv</td>
<td>33</td>
<td>0.721</td>
<td>89.3</td>
<td>92.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>65</td>
<td>2.94</td>
<td>91.4</td>
<td>93.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>129</td>
<td>12.0</td>
<td>93.2</td>
<td>95.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>257</td>
<td>48.7</td>
<td>93.7</td>
<td>96.5</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Uni-Node Support Efficiencies on SEMi Simulation of EARTH-MANNA
Results after simulation: Fibonnaci

Figure: EARTH-MANNA-S

Figure: EARTH-MANNA-D
Results after simulation: N-Queens-P

Figure: EARTH-MANNA-S

Figure: EARTH-MANNA-D
Results after simulation: N-Queens-T

Figure: EARTH-MANNA-S

Figure: EARTH-MANNA-D
Results after simulation: Paraffins

Figure: EARTH-MANNA-S

Figure: EARTH-MANNA-D
Results after simulation: Tomcatvv

Figure: EARTH-MANNA-S

Figure: EARTH-MANNA-D
What to Take Home (for now!)

Decide What to Model

- Communication?
- Context-switch?
- Latency vs throughput
- etc.

Decide How to Model

- Analytical
- Real measurements on (imperfect) hardware
- Simulation of enhancements to make to the HW

Define a Set of Benchmarks

- Microbenchmarks: must evaluate (verify) the quality of the PXM implementation
- Application benchmarks: must be representative (validate) of the workloads the PXM is supposed to help process
Bibliography


J. P. Buzen.
Fundamental laws of computer system performance.

L. Dagum and R. Menon.
Openmp: an industry standard api for shared-memory programming.

Evaluation of openmp task scheduling strategies.

J. Hoeflinger and B. de Supinski.
The openmp memory model.


Bibliography IV


