How’s the Parallel Computing Revolution Going?

Towards Parallel, Scalable VM Services

Kathryn S McKinley
The University of Texas at Austin
20th Century
Simplistic Hardware View

Faster
Processors
Frequency Scaling
Speculation, OO

programs do not change
they just run faster
Programming Language Evolution

Native Programming Languages

Managed Programming Languages
20th Century Simplistic Software View

hardware does not change

it just runs faster
Processor Technology Evolution

- Pentium 4 NetBurst (130nm) 2003
- Pentium M Dothan (90nm) 2005
- Power 5 (2 cores) (90nm) 2004
- Core 2 Duo Conroe (65nm) 2006
- Core 2 Duo Wolfdale (45nm) 2009
- Atom Diamondville (45nm) 2008
- Core 2 Duo Wolfdale (45nm) 2009
- i5 Clarkdale (32nm) 2010
The 20\textsuperscript{th} Century Virtuous Cycle

- Faster Single Processor Frequency Scaling
- Larger, More Capable Software Managed Languages
The 21\textsuperscript{st} Century Virtuous Cycle?

More Cores
\textit{Chip Multiprocessors} 
\textit{CMP}

Scalable Software
\textit{Scalable Apps + Scalable Runtime}
How is this new virtuous cycle going?
Measured Power vs Performance

SPEC CPU 2006, DaCapo, SPEC jvm98
How is this new virtuous cycle going for single threaded Java
Performance Scaling
Single Threaded Java Benchmarks
Core i7: 4 cores, 2 way SMT
How is this new virtuous cycle going for multi-threaded Java
Performance Scaling
Multi-Threaded Java Benchmarks
Core i7: 4 cores, 2 way SMT

Hardware Contexts

Speedup

avrora
batik
eclipse
h2
jython
luindex
lusearch
mtrt
pjbb2005
sunflow
tomcat
tradebeans
tradesoap
xalan
geomean

Kathryn McKinley
Towards Parallel, Scalable VM Services
Power, Performance, and Concurrency

- **Single threaded hollow; multithreaded solid**
- Microarchitecture changes from Pentium 4 (130) to i5 (32) favored parallelism-no surprise
- Multithreaded performance incurs a significant power cost
Is there hope?
Managed Languages

Challenges & Opportunities
Must Start with a Scalable Managed Runtime
Sequential Managed Programs

- Single Core
  - Application
  - Managed Runtime

- time

- Profiling
- Dynamic Analysis
- Compilation
- Garbage Collection
- Other Helper Threads
- ……
Steps towards scalability

**Step 1. Parallel application**

- Core 0
- Core 1
- Core 2
- Core 3
- Core 4
- Core 5
- Core 6
- Core 7

Each thread has different running time
Step 2. Parallel runtime

Runtime waits for all application threads to pause
Steps towards scalability

**Step 3. Parallel & concurrent runtime**

Managed runtime on application’s critical path may perturb its performance
Steps towards scalability
Ideal model

Step 4. Minimize perturbation

Application offloads work to concurrent runtime threads

Whole runtime task taken off critical path
Steps towards scalability
Ideal model

Step 4. Minimize perturbation

Worst case is parallel & concurrent
Vision

- **Scalable Runtimes**
  - Runtime & application parallelism & concurrency
  - CMP aware runtime improves application scalability

- **Communication**
  - Cache coherency is expensive and performance sensitive
  - Memory bandwidth scaling is problematic

- **Heterogeneity**
  - Move non-critical path off power-hungry cores
  - Smarter, more aggressive analysis

- **Specialization?**
  - Tuned cores? Special purpose cores?
Approach

- **Profiling** (feedback directed optimization)
  - Concurrent analysis
  - More invasive analysis on low-power cores

- **GC**
  - High performance concurrent GC
  - High performance non-moving GC
  - Reduced synchronization overheads
  - Distributed & scratchpad GC

- **JIT**
  - Concurrent, parallel JIT
  - Cost-benefit shift as low-power cores used

- **Architecture**
  - Tuned and/or specialized cores for runtime services
  - Coherence tailed for restricted, common case of GC
Today

- **Profiling** (feedback directed optimization)
  - Concurrent analysis
  - More invasive analysis on low-power cores
- **GC**
  - High performance concurrent GC
  - High performance non-moving GC
  - Reduced synchronization overheads
  - Distributed & scratchpad GC
- **JIT**
  - Concurrent, parallel JIT
  - Cost-benefit shift as low-power cores used
- **Architecture**
  - Tuned and/or specialized cores for runtime services
  - Coherence tailed for restricted, common case of GC
A Concurrent Dynamic Analysis Framework For CMP Hardware

Jungwoo Ha
U. Texas & UCS/ICI-East

Matthew Arnold
IBM Research

Stephen M. Blackburn
Australian National University

Kathryn S. McKinley
University of Texas at Austin

Kathryn McKinley
Towards Parallel, Scalable VM Services

27
Generic Sequential Analysis

- Difficult to optimize instrumented code
- Trade accuracy for overhead (sampling)
Generic Concurrent Analysis

- Lower overhead & higher accuracy
- Must deal with microarchitectural side-effects
Side-effects to Avoid

Core A  L1  lower level cache(s)  L1  Core B

Application (Producer)  Analysis (Consumer)

false & true sharing
High latency memory operation
Cache line ping-ponging
Cache-friendly Asymmetric Buffering

- **Lock-free** communication channel between application and analysis thread
- Cache-friendly asymmetric buffering
  - Actively avoids microarchitectural side-effects
  - **Enqueue**
    - light-weight instrumentation
    - produces one record at time
  - **Dequeue**
    - consumes one chunk (fraction of a buffer) at a time
Cache-friendly Asymmetric Buffering

- 16 slots on the buffer
- 4 chunks, 4 slots on each chunk
- L1 size == chunk size
### Cache-friendly Asymmetric Buffering

- Delay consumer dequeue operation until cache line is flushed
  - 2 chunks away (smiley location)
- Analyzer operates one chunk at a time
  - chunk_size > L1 size
  - In practice, chunk_size >= 2 * L1 works well.
Cache-friendly Asymmetric Buffering

- application blocks only when buffer is full
  - waiting until two more chunks are available
Cache-friendly Asymmetric Buffering

**Producer (sees buffer)**

```c
while (*bufptr != 0) {
    if (*bufptr == MAGIC)
        bufptr = buffer;
    if (*bufptr != 0)
        block();
}
*bufptr++ = data;
```

**Consumer (sees chunk)**

```c
while (app_is_running) {
    index = index_of(chunk_num+2);
    while (buffer[index] == 0)
        spin_or_sleep();
    consume(chunk_num);
    chunk_num = NEXT(chunk_num);
}
```

- producer may spin on `bufptr`, while consumer may spin on `buffer`
- Producer code common case is 6 instructions in x86.

Consumer only spins here

Kathryn McKinley
Towards Parallel, Scalable VM Services
Framework Provides ...

• Cache-friendly Asymmetric Buffering (CAB)
  – Minimizes microarchitectural side-effects
  – Quickly offloads event data from application’s critical path
• Configurable parameters for optimization
  – buffer size & chunk size
• Various collection mode
  – Exhaustive mode
  – Sampling mode
• Works on various threading model
  – N:M (green) threading model
  – native threading model
Evaluation

• 3 different CMP processors

Pentium 4 w/ hyperthreading

Core 2 Quad

Core i7
Evaluation

• Jikes RVM (2 different threading models)
  – N:M threading (Jikes RVM 2.9.2)
  – Native threading (Jikes RVM 3.0.1)
• Reference Dynamic Analysis Implementation
  – Method counting
  – Call graph
  – Call tree profiling
  – Path profiling
  – Cache simulator using load/store events
• Benchmarks
  – DaCapo, SPEC JVM 98 benchmark suites
• Parameters
  – buffer size = 2MB, chunk size = 128KB
Call Graph Profiling

- **Instrumentation Overhead** – *Bar 1*
- Bar1 – Collect event data and write into a single word. No analysis thread.

Kathryn McKinley  
Towards Parallel, Scalable VM Services  
39
Enqueueing Overhead – (Bar2 - Bar 1)

Bar2 – Collect event data and write into the buffer. No analysis thread
Call Graph Profiling

- **Communication Overhead** – \((\text{Bar3} - \text{Bar 2})\)
- **Bar3** – Analysis thread dequeues and write it into a single word.
Call Graph Profiling

- Analysis (data processing) Overhead = (Bar 5 – Bar 1)
- Bar4 – Concurrent Analysis
- Bar5 – Sequential Analysis
Call Graph Profiling

- **Overhead reduction with Concurrent Analysis** – \((\text{Bar 5} - \text{Bar 4})\)
- Bar4 – Concurrent Analysis
- Bar5 – Sequential Analysis
Path Profiling

- Overhead reduction with Concurrent Analysis – (Bar 5 – Bar 4)
- More data & computation than call graph
Path Profiling
Multithreaded Benchmarks

- Core 2
- Multi-threaded benchmarks
Concurrent Dynamic Analysis Framework

Conclusions

• Framework eases implementation of many client analyses
• CAB efficiently transfers application data to analysis thread avoiding microarchitectural side-effects
• Framework efficiently utilizes extra cycles to perform dynamic analysis concurrently
Related Work

• Concurrent Lock-free Queue
  – FastForward – single-producer & single-consumer [Giacomoni et al. 09]

• Concurrent analysis for specific clients
  – PiPA – cache simulator [Zhao et al. 08]

• Shadow process approach
  – Shadow profiling [Moseley et al. 07]
  – SuperPin [Wallace et al. 07]
Are we finished with CMP efficient buffering?
Are we finished with CMP efficient buffering?

Not yet
Are we finished with CMP efficient buffering?

Not yet

parallel analysis
memory scalable
self-tuning parameters
There is some hope

but we need many such base mechanisms
Software Challenges and Opportunities

Communication (efficient coherency)
Analysis (off critical path, new analyses)
GC (concurrent, parallel, high throughput)
JIT (concurrent, parallel, more aggressive)
Heterogeneity (exploit it)
Memory (PCM, bandwidth limits)
Hardware Challenges and Opportunities

Heterogeneity
- Tune cores to ubiquitous loads?
- Specialize for ubiquitous loads?

Coherence
- SMT coherency does not scale
- Software guarantees for simplified protocols?

Memory/Cache
- Exploit access behavior of managed languages
The 21\textsuperscript{st} Century Virtuous Cycle?

Questions?

More Cores
\textit{Chip Multiprocessors CMP}

Scalable Software
\textit{Scalable Apps + Scalable Runtime}

Kathryn McKinley
Towards Parallel, Scalable VM Services 54
I love my job

• Because when I fail, I get better
Failures

- Rejected: jobs (all)
- Failed: my Rice PhD qualifying exam
- Rejected: jobs (8 of 11)
- Rejected: my first three grant applications
- Bad teaching evaluations
- Rejected 2 times: my most cited paper
- Rejected: jobs (some)
- Rejected: papers, grants, papers, grants, ...
Processor Technologies and Power

- Thermal Design Power (TDP) or chip power budget
  - The amount of power consumed without exceeding the maximum junction temperature
- Power measurement
  - Hall effect current sensor on 12V line driving the chip
  - Sampling rate 50Hz

<table>
<thead>
<tr>
<th>Processor</th>
<th>µArch</th>
<th>µProcessor</th>
<th>Process nm</th>
<th># of Cores</th>
<th># of Threads</th>
<th>Clock GHz</th>
<th>LLC MB</th>
<th>TDP W</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium 4</td>
<td>NetBurst</td>
<td>Northwood</td>
<td>130</td>
<td>1</td>
<td>2</td>
<td>2.40</td>
<td>0.5</td>
<td>66.2</td>
<td>May ‘03</td>
</tr>
<tr>
<td>Core 2 Duo E6600</td>
<td>Core</td>
<td>Conroe</td>
<td>65</td>
<td>2</td>
<td>2</td>
<td>2.40</td>
<td>4.0</td>
<td>65.0</td>
<td>Jul ‘06</td>
</tr>
<tr>
<td>Core 2 Quad Q6600</td>
<td>Core</td>
<td>Kentsfield</td>
<td>65</td>
<td>4</td>
<td>4</td>
<td>2.40</td>
<td>8.0</td>
<td>105.0</td>
<td>Jan ‘07</td>
</tr>
<tr>
<td>Core i7 920</td>
<td>Nehalem</td>
<td>Bloomfield</td>
<td>45</td>
<td>4</td>
<td>8</td>
<td>2.66</td>
<td>8.0</td>
<td>130.0</td>
<td>Nov ‘08</td>
</tr>
<tr>
<td>Atom 230</td>
<td>Atom</td>
<td>Diamondville</td>
<td>45</td>
<td>1</td>
<td>2</td>
<td>1.66</td>
<td>0.5</td>
<td>4.0</td>
<td>Jun ‘08</td>
</tr>
<tr>
<td>Core 2 Duo E7600</td>
<td>Core</td>
<td>Wolfdale</td>
<td>45</td>
<td>2</td>
<td>2</td>
<td>3.06</td>
<td>3.0</td>
<td>65.0</td>
<td>May ‘09</td>
</tr>
<tr>
<td>Atom D510</td>
<td>Atom</td>
<td>Pineview</td>
<td>45</td>
<td>2</td>
<td>4</td>
<td>1.66</td>
<td>1.0</td>
<td>13.0</td>
<td>Dec ‘09</td>
</tr>
<tr>
<td>Core i5 670</td>
<td>Nehalem</td>
<td>Clarkdale</td>
<td>32</td>
<td>2</td>
<td>4</td>
<td>3.40</td>
<td>4.0</td>
<td>73.0</td>
<td>Jan ‘10</td>
</tr>
</tbody>
</table>
Native and Java Benchmarks

- 61 benchmarks from six suites
- Native (Fortran/C/C++) single-threaded (NST)
  - SPEC CINT2006 (12)
  - SPEC CFP2006 (15)
- Native multithreaded benchmark (NMT)
  - PARSEC (11)
- Java single-threaded benchmarks (JST)
  - SPEC JVM98 (6)
  - DaCapo-2006-10-MR2 (2)
  - DaCapo-9.12 (2)
- Java multithreaded benchmarks (JMT)
  - SPEC JVM98 (1)
  - DaCapo-9.12 (11)
  - PJBB2005 (1)
Compiler, JVMs, OS, and Performance

- Intel compiler v11.1 with -O3 for NST
- Gnu gcc compiler v4.4.1 with -O3 for NMT
- Java virtual machines
  - Sun (Oracle) HotSpot build 16.3-b01
  - Oracle JRockit build R28.0.0-679-130297
  - IBM J9 build pxi3260sr8
  - We measure and report the 5th iteration
- Operating system
  - 32-bit Ubuntu 9.10 Karmic
  - Linux kernel version 2.6.31
- Performance
  - Normalized execution time to Atom *Diamondville* (45nm)
  - Java with HotSpot default