Graphite Introduction and Overview

Goals, Architecture, and Performance
The Future of Multicore

- Computing has moved aggressively to multicore
- Up to 100 cores available this year
- 1000-core chips by 2016, if trends continue
- Are you ready?
Simulation in Multicore Research

• Simulation is vital for exploring future architectures
  – Experiment with new designs/technologies
  – Abstract away details and focus on key elements
  – Rapid exploration of design space
  – Early software development for upcoming architectures

• The future of multicore simulation:
  – Need to simulate 100’s to 1000’s of cores
  – Massive quantities of computation
  – High-level architecture is becoming more important than microarchitecture
    • On-chip networks, Memory hierarchies, DRAM access, Cache coherence
Graphite At-a-Glance

• A fast, high-level simulator for large-scale multicores

• Application-level simulation where threads are mapped to target cores

• Runs in parallel on multicore host machines

• Multi-machine distribution
  – Invisible to application
  – Runs off-the-shelf pthread apps

• Relaxed synchronization scheme
  – Trades some timing accuracy for performance
  – Guarantees functional correctness
Graphite Performance

Graphite slowdown on 8 host machines (64 cores total) versus native execution on one 8-core machine

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>41x</td>
</tr>
<tr>
<td>Max</td>
<td>3930x</td>
</tr>
<tr>
<td>Median</td>
<td>616x</td>
</tr>
</tbody>
</table>

- Typical slowdown for existing sequential simulators: 10,000x – 100,000x
- Results from SPLASH2 benchmarks on a 32-core target processor
Graphite Trades Accuracy for Performance

• Simulator performance is a major limiting factor
  – Limits depth and breadth of studies, size of benchmarks
  – Too much detail slows simulation
  – Cannot simulate 1000’s of cores
  – Most simulators are sequential, Graphite is parallel
  – Typical performance: 10,000x – 100,000x slowdown
  – Our target performance: 100x

• Performance vs. accuracy
  – Cycle-accurate: very accurate but slow
  – High-level: trade some accuracy for performance
  – For next year’s chips, you need cycle-accuracy
  – For chips 5-10 years out, you need performance
Outline

• Introduction
• Graphite Architecture
  – Overview
  – Multi-machine distribution
  – Clock Synchronization
• Results
• Conclusions
Graphite Overview

• Application-level simulator based on dynamic binary translation
  – Uses Intel’s Pin
  – App runs natively except for new features and modeled events
  – On trap, model functionality and timing

• Simulation consists of running an application on a target architecture
  – Target specified by swappable models and runtime parameters
    • Different architectures
    • Accuracy vs. Performance
  – Result:
    • Application output
    • Simulated time to completion
    • Statistics about processor events
Graphite Architecture

- Application threads mapped to target cores
  - On trap, use correct target core’s models

- Target cores are distributed among host processes

- Processes can be distributed to multiple host machines
Simulated Target Architecture

- Swappable models for processor, network, and memory hierarchy components
  - Explore different architectures
  - Trade accuracy for performance
- Cores may be homogeneous or heterogeneous
Key Simulator Components

- Host Machine
- MCP
- LCP
- Target Core

- Host Machine
- LCP
- Target Core

- Host Machine
- LCP
- Target Core

- Application Thread
- Messaging API
- Memory System
- Target Network Model
- Transport Layer

Physical Transport
Graphite implements a layered communication stack.

The application thread communicates with other threads via messages.
- Graphite messaging API
- Simulated shared memory

Messages are routed and timed by target architecture network model.

Transport layer delivers messages to destination target core.
- Host shared memory (same host process)
- TCP/IP (different host processes)
• Introduction

• Graphite Architecture
  – Overview
  – Multi-machine distribution
  – Clock Synchronization

• Results

• Conclusions
Parallel Distribution Challenges

• Wanted support for standard pthreads model
  – Allows use of off-the-shelf apps
  – Simulate coherent-shared-memory architectures

• Must provide the illusion that all threads are running in a single process on a single machine
  – Single shared address space
  – Thread spawning
  – System calls
Single Shared Address Space

- All application threads run in a single simulated address space
- Memory subsystem provides modeling as well as functionality
- Functionality implemented as part of the target memory models
  - Eliminate redundant work
  - Test correctness of memory models
Thread Distribution

- Graphite runs application threads across several host machines
- Must initialize each host process correctly
- Threads are automatically distributed by trapping threading calls
System Calls

• Three kinds of system calls need to be handled specially
  – System calls that pass memory operands to the kernel
  – System calls that implement synchronization/communication between threads
  – System calls that deal with allocating and deallocating dynamic memory

• Other system calls can simply be allowed to fall through
Outline

• Introduction
• Graphite Architecture
  – Overview
  – Multi-machine distribution
  – Clock Synchronization
• Results
• Conclusions
Clock Synchronization

- Cores *only* interact through messages
- Clocks are updated with message timestamps
Clock Synchronization

• Threads may run at different speeds, causing clocks to deviate
  – Clocks are only used for timing, functional correctness is always preserved
  – Must be synchronized on explicit interaction
  – Clocks may differ on implicit interaction → timing inaccuracy

• Define synchronization as *managing the skew of different target core clocks*.
  – This is *not* application synchronization!

• Graphite supports three synchronization schemes with different accuracy and performance tradeoffs
Synchronization Schemes

- **Lax**
  - Relies *exclusively* on application synchronization events to synchronize tiles’ local clocks
  - Functionally, events may occur out-of-order w.r.t. simulated time
  - Best performance; worst accuracy

- **LaxP2P**
  - Observation: Timing inaccuracy is due to a few outliers
  - Every \( N \) cycles, each target core randomly pairs with another
  - If cycles differ by too much, ‘future core’ goes to sleep
  - Good performance; good accuracy

- **LaxBar**
  - Every \( N \) cycles, all target cores wait on a barrier
  - Keeps cores tightly synchronized, imitates cycle-accuracy
  - Worst performance; best accuracy
Example Simulation (Lax)

- Application
- Synchronization Point
- App Exit
- Lax

Real time

Simulated time

Core 1
Core 2
Core 3
Example Simulation (LaxBar)

- App Exit
- Lax
- LaxP2P
- LaxBar
- Application Synchronization Point
- Barrier

Simulated time vs. Real time
Clock Skew Measurements

- Graphs show clock skew for each scheme using the fmm benchmark
  - Clock skew is the spread between minimum and maximum clocks at any given point
  - Note: Spikes on graphs due to errors in measurement method
- Lax has largest skew (~2,000,000 cycles)
  - Application synchronization events are clearly visible
  - Fine-grain thread interactions can be missed or misrepresented
- Lax P2P has much lower skew (~30,000 cycles)
  - Application synchronization events slightly visible
- LaxBar has low, constant skew (~4000 cycles)
Outline

• Introduction
• Graphite Architecture
• Results
  – Experimental methodology
  – Simulator performance and scaling
  – Synchronization scheme comparison
• Conclusions
• Matrix-multiply kernel running on **1024-core** target
• Simulator speed-up almost linear
  – 3.85x going from 1 to 10 host machines
Experimental Methodology

• Target Architecture:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of cores</td>
<td>32</td>
</tr>
<tr>
<td>L1 caches</td>
<td>Private, 32 kB per tile</td>
</tr>
<tr>
<td>L2 caches</td>
<td>Private, 3 MB per tile</td>
</tr>
<tr>
<td>Cache coherence scheme</td>
<td>Full-map directory based</td>
</tr>
<tr>
<td>Interconnection Network</td>
<td>2-D mesh</td>
</tr>
</tbody>
</table>

• SPLASH-2 benchmark suite

• All experimental results collected on 8-core Xeon host machines running Linux
Performance Scaling

- Graphite scales if the application scales
- Even non-ideal speedup still reduces latency and design iteration time
Performance Summary

- Sequential simulator slowdown is unacceptable
- Slowdown versus native execution as low as 41x
  - Would continue to drop with larger targets and more hosts
- Simulator overhead depends heavily on application characteristics
- Still more room for optimization

<table>
<thead>
<tr>
<th></th>
<th>Slowdown over native execution on 8 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential (1 core)</td>
</tr>
<tr>
<td>Min</td>
<td>580x</td>
</tr>
<tr>
<td>Max</td>
<td>17,459x</td>
</tr>
<tr>
<td>Mean</td>
<td>8,027x</td>
</tr>
<tr>
<td>Median</td>
<td>6,940x</td>
</tr>
</tbody>
</table>

* Host machines are 8-core servers
Synchronization Results

Performance

- Normalized to Lax on a single host
- Trends
  - Lax gives best performance
  - LaxP2P is nearly as good
  - LaxBar is significantly worse

Accuracy

- %-deviation from LaxBar on 1 machine
- Trends
  - Lax shows high deviation
  - LaxP2P and LaxBar show generally low deviation
Lax Synchronization Results

LaxBar on 1 host machine

Mean

Std. Dev.

- Benchmark

Lax

LaxP2P

LaxBar

Performance

Deviation (%)

LaxBar on 1 host machine

32
Summary

• Graphite accelerates multicore simulation using multi-machine parallel distribution
  – Enables simulation of 1000’s of cores
  – Invisible to application, runs off-the-shelf pthread apps

• Graphite provides fast, scalable performance
  – As little as 41x slowdown vs. native execution
  – Up to 20x speedup on 64 host cores (across 8 machines)

• LaxP2P synchronization provides a good balance between performance and accuracy