Trace-driven simulation of multithreaded applications in gem5
Gilles Sassatelli, Alejandro Nocua, Florent Bruguier, Anastasiia Butko

To cite this version:
Trace-driven simulation of multithreaded applications in gem5

Gilles SASSATELLI
sassatelli@lirmm.fr

Alejandro NOCUA, Florent BRUGUIER, Anastasiia BUTKO
Cambridge, UK – September 11, 2017
Mont-Blanc 1, 2 & 3 projects (FP7, H2020)
- Getting ARM technology ready for HPC: HW, SW & Apps
- Advances in energy efficiency towards Exascale

Initial effort: using gem5 for performance prediction (2011)
- STE Nova A9500 SoC (dual-core Cortex A9)
- Fed publicly available parameters into a gem5 FS model
- 1.5% - 18% error, due to rough DRAM model and interconnect

Background motivations

Calibration against real hardware
- Using gem5 for performance prediction
- And McPAT for power estimation

Then onto exploring fancy architecture configurations
- Heterogeneous single-ISA multicores à la big.LITTLE
- Assymetric, 3 levels of heterogeneity etc.
Background motivations cont’d

- Not ready for manycores, too slow!
  - 1K-1M (simulated) IPS
  - Scales bad with system size
  - Already much better than RTL though

- Trading speed for accuracy? Any sweet spot?

![Diagram showing trade-off between accuracy and speed](image-url)
Background motivations cont’d

- Not ready for manycores, too slow!
  - 1K-1M (simulated) IPS
  - Scales bad with system size
  - Already much better than RTL though

- Trading speed for accuracy?

![Diagram showing trade-off between accuracy and speed for different simulation approaches such as RTL, Cycle-accurate, Cycle-approximate, Distributed, Trace-driven, and JIT.]
**Trace-driven simulation principle**

- ~ 70% simulation effort goes into CPU
  - Abstracting away CPU cores sounds like a good idea
  - Between 2 consecutive L1 cache misses (in-order) CPU cores perform « consistently »
SimMATE: 2-stage process

- Trace collection: tracing only L1 miss related transactions
- Trace replay: Using trace injectors that initiate transactions as previously recorded
SimMATE for faster DSE

- **Trace collection = freezing**
  - CPU parameters alongside application SW
  - Private caches sizes, speed etc.

- **Trace replay allows exploring the rest**
  - L2 size, policy etc.
  - Interconnect type & speed
  - Main memory speed

---

**Trace Collection**

- Core 0
- Core n
- Interconnect
- Memory

**Trace Simulation**

- TI 0
- TI n
- Interconnect
- Memory
- L2

1 time

Multiple times
Trace replay performs event (re-) scheduling

- Simple « time shifting » approach
- Maintaining constant compute phases
SimMATE Benchmarking

- **Tuning DRAM latency**
  - Collection performed with 30ns
  - TD simulation from 5ns to 55ns
  - FS used as reference

![Diagram showing execution time error in percentage for TD and FS simulations with different latencies.](image)

**Execution time error [%]**

<table>
<thead>
<tr>
<th>Latency</th>
<th>MJPEG</th>
<th>FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5ns</td>
<td>2.8%</td>
<td>5.8%</td>
</tr>
<tr>
<td>15ns</td>
<td>4.5%</td>
<td>5.1%</td>
</tr>
<tr>
<td>30ns</td>
<td>0.1%</td>
<td>5.1%</td>
</tr>
<tr>
<td>45ns</td>
<td>0.4%</td>
<td>6.0%</td>
</tr>
<tr>
<td>55ns</td>
<td>0.4%</td>
<td>5.8%</td>
</tr>
</tbody>
</table>

Original
Tuning L2 size

- Collection performed without L2
- TD simulation from 0 to 16MB L2
- Errors originate from Cold-start bias / cache warmup

![Graph showing execution time error (%) for different L2 cache sizes](image)
Having these traces collected makes it easy to:

- Perform « Trace replication » i.e. emulate more CPU cores for scalability study
- This corresponds to weak scaling experiments, i.e. per-core workload remains same
Yet synchronizations must be accounted for!

- Using whatever API: POSIX threads, OpenMP 3.0 …
- Approach: embed synchronizations into traces
- Have an arbiter that takes care of locking (when barrier reached) and unlocking TIs
And most ARM AP are OoO (Out-of-Order)
  - Meaning multiple outstanding memory transactions
  - The assumption of constant time btw. 2 misses does not hold

big-LITTLE & other heterogeneous friends everywhere
  - And there microarchitecture details cannot be overlooked
Modeling micro-architecture timing & dependencies

- Tracing with O3 model + probes, without L2 cache
- Replay done in a smart «elastic» fashion

Elastic Traces: Trace-driven simulation for OoO

http://gem5.org/TraceCPU
Smart TraceCPU

- Updating a dependency graph pushing ready instructions into a queue for issue
SimMATE + Elastic Traces = ElasticSimMATE

- Enabling both OoO + multithreaded applications
- Key: embed synchronization information @ tracing time.

Barriers etc.

Elastic Trace

Instruction trace
- Header:
- Req:

Synchronization trace
- ... PC Event Inst Data
- ...

Data dependency trace
- Header:
- Inst:
- ...

Trace CPU

I-side generator

Synchro generator

D-side generator

Arbiter

Global Stop/go
Proper tracing of synchronizations

- API-dependant: OpenMP 3.x
- Tracing whenever entering or leaving parallel region, barrier etc.

```c
#pragma omp parallel
for(i=0;i<n;i++) {
    /* do_some work */
}
```

```c
for(i=0;i<n;i++) {
    OMP_runtime_call()
    /* do_some work */
}
```

```
Tick    PC  th_id  type  IC     DC
389178  177216 0  1      1081157 165738
...
```
ESW wrapup

- ESM flow wrapup
  - Using BSC Mercurium compiler / Nanos++ runtime
  - Tweaked runtime such that custom m5 pseudo instructions produce trace records

```
#pragma omp parallel
for(i=0;i<n;i++) {
    /* do_something */
}
```

Unmodified sources

**gem5**

Trace Collection

App Binaries

```
#pragma omp parallel
for(i=0;i<n;i++)
    nanos_create_team();
/* do_something */
    nanos_end_team();
}
```

Nanos++ runtime

Ref_Architecture

Generated Traces

Instruction

Dependency

Synchronization

gem5

Trace Replay

Target_Arch 1
Two main use cases:

- Fast parameter exploration
- Scalability study: « trace replication »

Speedup & accuracy?

- Experiments on Rodinia application kernels
Benchmarks

- Two main use cases:
  - Fast parameter exploration
  - Scalability study: « trace replication »

- Speedup & accuracy?
  - Experiments on Rodinia application kernels

![Graphs showing execution time and error percentage for different L2 cache sizes.](image)
Benchmarking

- Two main use cases:
  - Fast parameter exploration
  - Scalability study: « trace replication »

- Speedup & accuracy?
  - Experiments on Rodinia application kernels

![Graph showing execution time and error percentage vs L2 cache size for different cache sizes and benchmarks.](image)
Benchmarking

- **Two main use cases:**
  - Fast parameter exploration
  - Scalability study: « trace replication »

- **Speedup & accuracy?**
  - Experiments on Rodinia application kernels

---

![L2 Cache Miss Rate][1]

![L2 Cache Overall Miss Latency][2]

---

**L2**

**KMEANS**
Benchmarks

- **Two main use cases:**
  - Fast parameter exploration
  - Scalability study: « trace replication »

- **Speedup & accuracy?**
  - Experiments on Rodinia application kernels

---

![Graphs showing execution and simulation time across varying number of cores.](image)

- **128 cores**
- **KMEANS**

---

![Diagram illustrating trace collection and simulation processes.](image)
Scalability analysis has limits
- Requires additional features s.a. address offsetting
- **Weak scaling** only (replicated per-core workloads)

Programming models moving from loops to *tasks*
- OpenMP 4.0, OmpSs
- Still pragma-based
- More parallelisms available at run-time … more opportunities for smart job scheduling

BSC OmpSs: Cholesky decomposition
Unbinding traces from cores

- One trace per Task, not per core!
- Assign traces to cores by emulating runtime behaviour in trace replay
- This is real strong scaling

Trace collection

Trace simulation

Scheduler / runtime (emulated)
- **Current ESM prototype**
  - ~5x - 10x speedup for low core count, probably more for tens / thousands
  - Nice solution for fast DSE
  - Remaining accuracy issues for some applications
    - Common to ESM & Elastic Traces
    - Under investigation with ARM
- **Use cases**
  - Exploration of memory subsystem
  - Some microarchitecture parameters (Elastic Traces)
  - …
- **Future directions**
  - Ruby compatibility
  - Could be combined with other initiatives (dist-gem5)
  - Can be extended to other PM / APIs (Tasking, MPI…)

http://montblanc-project.eu