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

To cite this version:


HAL Id: lirmm-01723755
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01723755

Submitted on 18 Jun 2019
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

Background motivations cont’d

- **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.

---

“Big” cluster

```
+------------------+
| A7               |
| A7               |
+------------------+
```

“LITTLE” cluster

```
+------------------+
| A7               |
| A7               |
+------------------+
```

Interconnect

```
| Interconnect      |
+-------------------|
|                  |
|                  |
|                  |
|                  |
+-------------------|
```

Memory

```
+------------------+
| Memory           |
| Memory           |
+------------------+
```

RAM

```
| RAM              |
| RAM              |
+------------------+
```

---

![](image1.png)

```
Execution time [Seconds]
```

```
Energy-to-Solution [Joules]
```

```
Baseline (4A7/4A15)
SMP
big.LITTLE/big.Medium
big.Medium.LITTLE
Energy Delay Product
Energy Delay Product
```

---

![](image2.png)
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)
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?

Background motivations cont’d
~ 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

**I 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 MJPEG and FFT across different latency times](image)

**Execution time error [%]**
- MJPEG
  - Original: 0.6%
  - 5ns: 2.8%
  - 15ns: 4.5%
  - 30ns: 5.8%
  - 45ns: 5.1%
  - 55ns: 6%
- FFT
  - Original: 0.5%
  - 5ns: 0.4%
  - 15ns: 0.4%
  - 30ns: 0.1%
  - 45ns: 0.4%
  - 55ns: 0.4%
SimMATE Benchmarking

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

![Execution time error chart]

<table>
<thead>
<tr>
<th>L2 cache size</th>
<th>ET</th>
<th>ET+init</th>
<th>ET+OS boot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>0.1%</td>
<td>8%</td>
<td>7.5%</td>
</tr>
<tr>
<td>256kB</td>
<td>15%</td>
<td>8%</td>
<td>7.5%</td>
</tr>
<tr>
<td>1MB</td>
<td>1%</td>
<td>0.3%</td>
<td>14%</td>
</tr>
<tr>
<td>8MB</td>
<td>18%</td>
<td>10%</td>
<td>15%</td>
</tr>
<tr>
<td>16MB</td>
<td>15%</td>
<td>8%</td>
<td>7%</td>
</tr>
</tbody>
</table>
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
Limitation: in-order only!

- 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

http://gem5.org/TraceCPU
Elastic Traces: Trace-driven simulation for OoO

- **Smart TraceCPU**
  - Updating a dependency graph pushing ready instructions into a queue for issue

- **Fixed I-side Generator**
  - `init()`:
    - Read trace
    - Schedule event for first request
  - `tryNext()`:
    - Send current request
    - Read next request
    - Schedule event for next `tryNext()`

- **Elastic D-side Generator**
  - `init()`:
    - Read trace & populate DepGraph
    - Add root node to the readyList
    - Schedule event for execute() at execute time of the head of the readyList
  - `completeMemAccess()`:
    - Update HW resources for store
    - Update DepGraph, child nodes for completed load
    - checkAndIssue() any new dependency-free nodes. Add nodes that fail to issue to depFreeQueue.
    - Schedule event for next execute()
  - `execute()`:
    - checkAndIssue() from depFreeQueue
    - Check HW resource availability and issue, i.e. add to readyList
    - Loop through nodes in readyList
      - If STORE, send request
        - STORE complete
      - If LOAD, send request
        - COMP complete
      - If COMP, do nothing
        - Update DepGraph, child nodes for completed nodes
        - checkAndIssue() any new dependency-free nodes. Add nodes that fail to issue to depFreeQueue.
        - Populate DepGraph by reading from trace
        - Schedule event for execute() at execute time of the head of the readyList

- **http://gem5.org/TraceCPU**
SimMATE + Elastic Traces = ElasticSimMATE

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

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

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

**Unmodified sources**

**gem5 Trace Collection**

**Generated Traces**

**gem5 Trace Replay**

**Thread creation & such m5_trace(TYPE,th_id)**
Two main use cases:

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

Speedup & accuracy?

- Experiments on Rodinia application kernels
Benchmarking

- **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 vs L2 cache size for various cache sizes including 0MB, 16kB, 128kB, 512kB, 1MB, and 2MB.]
Two main use cases:

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

Speedup & accuracy?

- Experiments on Rodinia application kernels
Two main use cases:

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

Speedup & accuracy?

- Experiments on Rodinia application kernels

![Graphs showing L2 Cache Miss Rate and Overall Miss Latency for different cache sizes and benchmarks.](chart.png)
Two main use cases:

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

Speedup & accuracy?

- Experiments on Rodinia application kernels

Benchmarking

128 cores  KMEANS
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 parallelims available at run-time … more opportunities for smart job scheduling
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

- Ready Task queue
- Scheduler
- T0, T1, T2, T3

### Trace simulation

- Task 1 trace
- Task 2 trace
- Task N trace
- Task dependencies

- Scheduler / runtime (emulated)
- Task 1 trace
- Task 2 trace
- Task 1 trace
- Task 1 trace
- Task 2 trace

- Interconnect
- Memory
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