A Trace-driven Approach for Fast and Accurate Simulation of Manycore Architectures

Anastasiia Butko, Rafael Garibotti, Luciano Ost,
Vianney Lapotre, Abdoulaye Gamatie and Gilles Sassatelli
LIRMM (CNRS and University of Montpellier 2), Montpellier, France
e-mail: {firstname.lastname}@lirmm.fr

Chris Adeniyi-Jones
ARM Ltd.,
Cambridge, United Kingdom
e-mail: Chris.Adeniyi-Jones@arm.com

Abstract— The evolution of manycore sytems, forecasted to feature hundreds of cores by the end of the decade calls for efficient solutions for design space exploration and debugging. Among the relevant existing solutions the well-known gem5 simulator provides a rich architecture description framework. However, these features come at the price of prohibitive simulation time that limits the scope of possible explorations to configurations made of tens of cores. To address this limitation, this paper proposes a novel trace-driven simulation approach for efficient exploration of manycore architectures.

I. Introduction

To achieve efficient exploration of emerging manycore systems, flexible and scalable simulators are mandatory. Such simulators should combine efficient modeling, debugging and simulation capabilities for verifying the software development, while meeting the expected functionality and performance objectives of the platform. Available industrial and academic simulators differ in terms of simulation speed/accuracy tradeoffs, and their adoption is usually defined by desired exploration level. Quasi-cycle accurate simulators [1] are popular and attractive for architectural exploration. The gem5 environment is a popular open-source quasi-cycle accurate simulator [2] [3] that supports a rich set of instruction set architectures (ISAs). It has an active development and support community.

Even though enabling flexible and detailed microarchitecture evaluation, gem5 entails slow simulation speeds, thereby limiting its scope of applicability to systems with hundreds of cores. This calls for alternative approaches capable of providing high simulation speed while preserving accuracy that is crucial to architectural exploration.

The presented work aims at proposing a novel scalable trace-driven simulation approach for event-driven computer architecture simulators such as gem5. The fundamental principle of the approach lies in decreasing simulation complexity by *abstracting away core execution into traces*, as follows:

1. core execution traces, i.e. incoming/outgoing memory transactions, are captured in a full-system simulation;
2. these traces are augmented with *synchronization semantics*, then *replicated* into so-called *augmented vector traces* to simulate systems made of a higher core count;
3. augmented vector traces are replayed into a final gem5 simulation through *traffic injectors*; only interconnect and memory subsystem are actually simulated thereby resulting in significant performance boost.

This approach is implemented in gem5 and validated on ARM ISA (it operates however on other ISAs). Known limitations lie in the trace-driven nature of the approach: threads are pinned to cores and run-time features such as load-balancing are not modeled any further. Our solution advances state-of-the-art in trace-driven simulation through its ability to accurately simulate a computer architecture made of M cores based on traces captured in a reference simulation on a system comprising N cores, with $M \geq N$, thanks to *trace replication*.

In the rest of this paper, Section II discusses related work. Section III provides the fundamental concepts of proposed trace-driven approach with its development and integration into gem5. Section IV gives an extensive evaluation of speedup, accuracy and cost of our simulation approach. Finally, Section V points out conclusions and future work.

II. Related work

Several techniques have been investigated for efficient system simulation, and we classify them in three main families. The first family concerns simulators relying on just-in-time (JIT) dynamic binary translation, e.g. OVP [4] and QEMU [5]. Such simulators can achieve speeds close to thousands MIPS [6] at the cost of limited accuracy. They often focus on functional validation rather than those of architectural exploration. However, some efforts to apply them to architecture exploration have been conducted in [7]. These approaches miss expressive modeling supports such as those related to cache hierarchies and coherence protocols.

The second family emphasizes the distribution of the simulation over multiple host machines [8]. Graphite [9] is a distributed simulator dealing with functional behavior. It minimizes synchronization overhead by abstracting away events ordering along the simulation. So, while decreasing simulation costs, such a relaxed synchronization vision limits architectural explorations such as communication bottlenecks. ZSIM [6] improves simulation speed by parallelizing the simulation on x86 multicore hosts. Authors claim about 2/3 and 4 orders of magnitude speedup than respectively Graphite and gem5. More generally, the use of distributed simulators is delicate in the sense that users have to carefully deal with simulation partitioning and synchronization among available CPUs, which limits its simulation speedup.

The third family concentrates on optimizing component descriptions (e.g. CPUs, interconnect infrastructure) following the transaction-level modelling (TLM) [10] or by using trace-

driven simulation [11]. While reducing the number of simulation events, the use of TLM for architecture exploration is strongly penalized by the lack of accurate microarchitecture modeling capabilities. Trace-driven modeling on the other hand, is a relevant approach in high-performance and embedded computing for reducing simulation cost. In [12], a trace-driven simulation for systems with 64-cores achieves in average 150x simulation speed w.r.t the reference Simics-based simulator [13]. Authors in [14] combine a network simulator with trace-based MPI simulator to explore the impact of interconnection network on the performance of parallel applications. In [15], trace-driven injectors enable to drive the optimization of networks-on-chip (NoCs) according to application traffic traces. Such approaches do not focus on microarchitecture issues. Another trace-driven simulator is TaskSim [11], dedicated to design space exploration of shared-memory multicore systems. It concentrates on multithreaded application behaviors instead of microarchitecture aspects.

Our contribution differs from all previous works by proposing a novel trace-driven simulation approach suitable for an efficient exploration of manycore architectures. Among its strengths are a source-level dependency management for thread synchronizations and a trace replication technique. It is capable of properly taking into account control and data dependencies of thread-oriented APIs such as POSIX threads and OpenMP.

III. Trace-Driven Approach

A. General Concept

The proposed trace-driven (TD) simulation approach is composed of three phases as shown in Fig.1: (a) *trace collection*, (b) *trace processing* and (c) *trace simulation*.

The first phase is devoted to trace collection. We define the target system as consisting of the hardware architecture and software system. In Fig.1 (a), the hardware architecture comprises N cores, each one having its private instruction (I$) and data (D$) caches. Communications between cores and external memory is achieved via an arbitrary communication infrastructure that may comprise caches (L2, L3). Software system includes the application code and the operating system. As modeled system are of shared-memory type all communications take place through memory locations that are being accessed by code executed on cores. Fig.1 depicts communications as a stream of *request* and *response* events via the communication and memory infrastructures. The structure of a trace contains

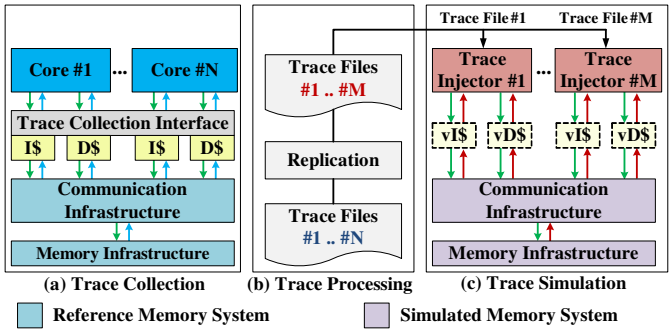

Fig. 1. Structural view of trace-driven simulation phases.

the memory access type (read or write), destination address, data value, the time at which a request is sent and satisfied. The *trace collection interface* (TCI) shown in Fig.1 (a) tracks and stores all request and response traces. It is located at the interface of each private cache memory. This allows to capture information on the core side memory requests and corresponding memory side responses. As a global output of collection phase a trace-set consisting of one trace-file for each core is obtained. Only trace information related to cache miss events are kept as only those result in external memory traffic.

The trace processing phase is shown in Fig.1 (b). Collected traces are augmented with synchronization information semantics which is automatically inserted from annotations at source-code level, this technique and also a trace-replication technique are explained in details in Sections III-B and IV-D.

The TD simulation phase depicted in Fig.1 (c) takes the augmented vector traces as inputs, and per-core traces are injected in the communication and memory subsystem by *trace injectors* (TIs). I$ and D$ are replaced by their virtual implementation (vI$ and vD$) to manage cache misses only. Given the output of our modular TD approach, one can analyse system behaviors by exploring a variety of parameters in communication and memory infrastructures. Parameters of cores are configured during trace collection phase.

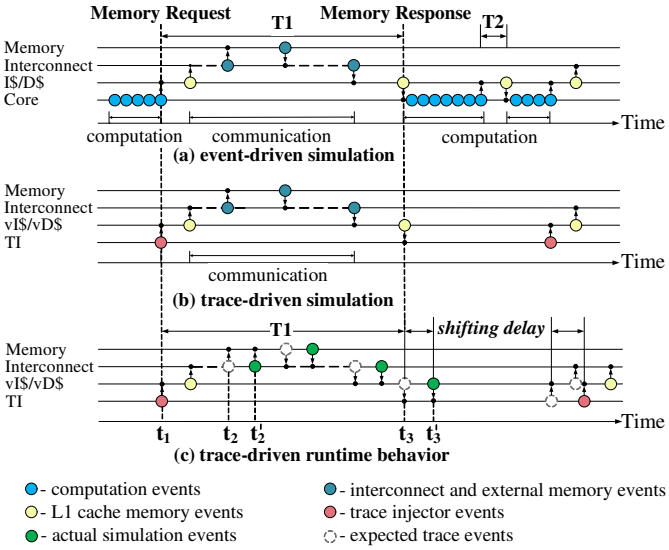

Fig. 2. Comparing event-driven and trace-driven simulations.

To identify the gain of the TD approach, we consider simulation process from two perspectives: full event-driven simulation and the corresponding TD simulation. These perspectives are illustrated in Fig.2, where a target architecture consists of four communicating layers corresponding to cores/TIs, private L1 cache memory, interconnection component and external memory. The time intervals T1 and T2 respectively represent the induced durations in case of cache miss and cache hit. According to the event-driven paradigm, we assume that the simulation time for both full event-driven simulation and trace-driven simulation is proportional to the number of simulated events. Thus, the gain in term of simulation time using the proposed TD approach depends on the number of events collected and filtered during the trace collection phase.

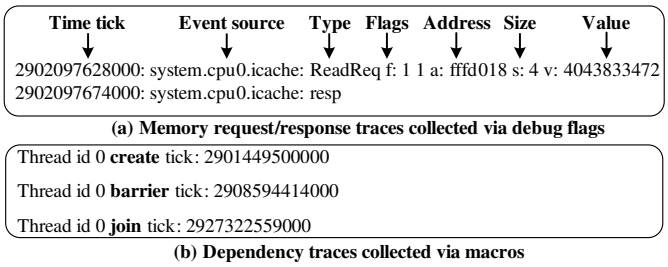

Fig. 3. An example of collected trace file extract.

Let us consider a reference trace collected according to Fig.1. Assuming the purpose lies in determining the impact of memory performance, memory responses are likely to occur at different time instants in the TD simulation compared to the recorded timeline in the reference simulation used for trace collection. This is shown in Fig.2 (c) in which a TI issues a request at time t_1, which reaches the memory at time t'_2, while in the reference simulation this request was expected at t_2 (represented by bullets with dashed borders). This implies a time shift that postpones 1) the response to the injector at time t'_3 instead of t_3, and 2) the next TI injection time. This is handled by processing time intervals between a response and the next request from the TI rather than absolute time.

B. Integration in gem5

The implementation of our TD simulation approach in gem5 follows the previous three phases.

Trace Collection and Processing. The first two phases are treated at the same time. Trace collection includes two important aspects: the hardware architecture and the software system. On the hardware structure side, to implement the TCI gem5 already has the necessary functionality that is part of its trace-based debugging which contains `DPRINTF` statements. It provides a flexible text-format file, which can be in some cases sub-optimal in terms of space and processing time. Fortunately, well-known optimization techniques such as coding and compression address this issue.

The gem5 memory subsystem is based on the notions of port, packet and request/response. Ports connect memory components to each other. Requests and responses are used to encapsulate CPU or I/O device messages [2]. Our implementation of trace collection interface relies on these notions. Fig.3 (a) shows an extract of the collected trace file.

On the software system side, we consider a multithreading programming model where multiple threads exist within the same process and share resources, e.g., memory. On a many-core system, every core executes a separate thread simultaneously. In order to deal with the resource sharing, threads are equipped with synchronization mechanisms. The synchronization points (barriers and join) introduce control-flow dependencies, addressed via a trace synchronization mechanism.

The first step of our synchronization mechanism consists in collecting additional synchronization traces. For that purpose, we add macros to input application source code so as to capture synchronization points. The gem5 specific operation `rpns()` returns the time at which it has been invoked in the format of internal time count, e.g. ticks. We create three macros, `CREATE`, `BARRIER` and `JOIN`, based on POSIX thread API and the `rpns()` operation of gem5. The execution output of an application annotated with such macros is presented in Fig.3 (b). This makes it possible to automatically append in the execution trace synchronization information that are later used.

Trace-Driven Simulation. For the simulation of the augmented vector traces, two main components are implemented in gem5 simulator: (i) *Trace Injector* and (ii) *Trace Arbiter*.

The TI consists of two fundamental modules: the first is dedicated to parse a trace file and to provide request/response structures, whereas the second triggers request injection and time interval management, i.e. time shifting of upcoming requests according to response times.

The second important component is trace arbiter. It is connected to each injector and has a global view of the entire system. Arbiter deals with dependency traces collected via the macros. Communications between the arbiter and injectors are achieved via a simple protocol. TI has two basic states: LOCK and UNLOCK. After initialization, it is unlocked and reads the trace file. As soon as the parser reaches a synchronization entry in the trace, an injector sends a corresponding signal to the arbiter. The arbiter sets the injector to LOCK state and waits until all the other injectors reach the same synchronization point. Once it happens, the locked arbiter unlocks all injectors and the simulation continues. Exchange of messages between the arbiter and the injectors is carried by atomic packets and does not affect the runtime. Arbiter and injector connections are shown in Fig.4.

One additional modification concerns the cache. Since the TD simulation starts with a cold (empty) memory it causes a form of bias. This problem is usually referred to as cold-start bias [16]. For private caches, we exploit the fact that all memory requests are cache misses and are declared as such for the TD simulation. This solution eliminates cold-start error, but brings a new restriction: each individual trace file collected through one cache size cannot be simulated with other cache sizes. For the other levels of cache, e.g. L2, there are a few methods, discussed in [16], reducing the problem.

One important trace simulation scenario is *trace replication*. The aim is to collect traces from N-core systems, then replicate them and simulate M-core systems, where M $>$ N. Such a replication is applicable when a given N-core system executes N-thread application where computation phases are time-bounded. In this case an application behavior pattern can be duplicated to simulate M-core system thus increasing the virtual size of application problem. A major feature of our technique is the direct access to the memory reference address field which allows the TI apply a smart memory mapping bypassing the compiler and the application source code.

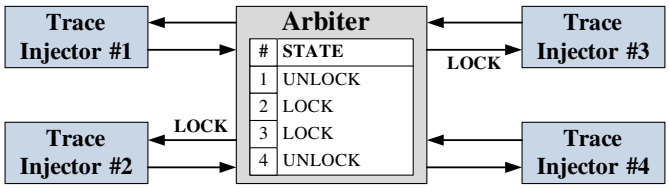

Fig. 4. Four TIs connection with an arbiter and two-sided communication of dependency management.

TABLE I
SUMMARY OF MAIN SIMULATION RESULTS: SPEEDUP, ACCURACY, COST IN TERMS OF MEMORY.

Applications		FIR	MJPEG	SW	Hist	Sort	N-body	VO	Reduct	FFT	Radix	LU	Ocean	Barnes
Simulation	**FS**	21.77	3.62	3.05	60.01	2306	78.38	23.45	97.82	119.57	3.61	16.61	15.92	10.35
time(seconds)	**TD**	1.03	0.03	0.03	0.073	6.28	0.62	0.72	0.14	14.08	0.11	0.43	2.82	0.52
Gain		21	136	122	800	366	79	33	734	8.5	36	38	6	20
Trace file (Gigabytes)		0.87	0.04	0.05	0.06	5.48	0.66	0.92	2.49	11.6	0.99	2.51	2.48	1.04
Execution	**FS**	0.4549	0.0264	0.0186	0.2672	48.169	0.8811	0.3010	0.3716	1.745	0.0188	0.4790	0.6027	0.1617
time (seconds)	**TD**	0.4546	0.0262	0.0178	0.2649	48.150	0.8809	0.3009	0.3715	1.852	0.0191	0.4843	0.6025	0.1613
Error (%)		0.07	0.98	4.26	0.85	0.04	0.03	0.02	0.02	5.79	1.84	1.09	0.03	0.21

Limitations. Our approach has three main constraints. The first is related to the core and L1 cache abstraction. It requires to recollect traces and change their configuration, e.g. core frequency, L1 cache size, etc. The second constraint concerns the nature of considered applications. Our trace replication approach does not support applications in which the computation phase cannot be statically bounded, e.g. data-dependent applications for which this phase depends on input data. The third constraint concerns simulation host machine capabilities. Indeed some traces could take tens of GB disk space and tens of GB operating memory for 512 cores simulation. Thus, considered machines should provide enough storage space.

IV. TRACE-DRIVEN APPROACH EVALUATION

We validate our TD approach by providing a detailed analysis of the simulation process. Our *reference platform* (RP) is characterized by a set of parameters: (i) 1-,2-,4-,8-cores processor (500 MHz), (ii) 4-kB private L1 D/I caches, (iii) 64 bits channel width, (iv) 30 ns DDR memory latency and (v) Linux Kernel 2.6.38. The ARM in-order ISA model of gem5 is used. The processors are connected to the DDR memory via a bus.

We consider a set of applications from scientific and multimedia computing domain, implemented in POSIX Threads. Some of them come from the SPLASH-2 benchmark suite [17]: Radix, Barnes, LU and Ocean, which are relevant due to the presence of multiple dependencies in the corresponding algorithms. In addition, we adopt further applications: Motion JPEG (MJPEG), Finite Impulse Filter (FIR), Smith Waterman (SW), Histogram for histogram graph computing, Merge Sort, N-body for simulating a dynamical system of particles, Reduction of vectors, Vector Operations (VO) and Fast Fourier Transform (FFT).

The considered RP is configured to run the above benchmarks in the $timing$ Full System (FS) mode of gem5. We address a thorough assessment of our TD approach by studying the gain in simulation speedup, accuracy and cost in terms of execution resources (mainly memory).

A. Evaluation of Speedup and Accuracy

Trace files are collected from the above RP with eight cores, and then executed in our TD simulator without any architectural change. Table I summarizes the simulation results. These comprise the simulation and application execution times (ETs) of both gem5 FS and TD simulation, the size of generated trace files, the gain in terms of speedup and the error percentage.

The results provided by the TD simulation show that the gain in terms of speedup is in a fairly wide range of 6x-800x depending on the application nature, e.g. computation-to-communication ratio. The size of obtained trace files and the error of TD simulation are respectively is within 40Mb – 11.6Gb and 0.02% – 5.79%.

B. Exploration of Architectural Parameters

We evaluate the TD simulation consistency by varying the internal architectural parameters just after the trace collection phase. Five memory latency values are used: 5ns, 15ns, 30ns, 45ns and 55ns. Their impact on two applications (MJPEG and FFT) execution time is evaluated, as illustrated in Fig.5. Results show that the error in terms of execution time is around 6%. Thus, our TD simulation reproduces the application behavior properly even when architectural parameters change.

In another experiment, we used the captured traces to evaluate system configurations including components that were not present during the original trace collection phase. To illustrate this evaluation, traces are collected from our RP with eight cores running MJPEG, and then transferred to a TD system, which includes an L2 cache shared between all TIs.

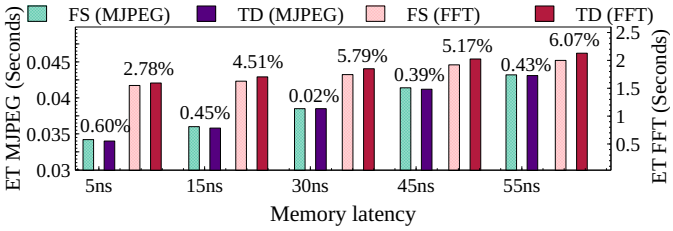

Fig. 5. Execution time and error comparing the gem5 FS vs. TD.

Here, in order to minimize the error in terms of execution time, we must address the cold-start bias issue as discussed in Section III-B. We propose to warm up L2 cache by considering traces captured before application execution phase. We collect and compare three traces: (i) execution time (ET) traces, (ii) ET with initialization phase traces and (iii) ET + initialization and OS boot phases. The results are presented in Fig.7. The traces in (i) do not provide the expected performance improvement and lead to 14.01% of absolute error on average. By using the traces in (ii), we observe a speedup and obtain an absolute error of 7.88% on average, which is 2 times less than provided by (i). The traces in (iii) give 6.60% of absolute error on average, which is the best.

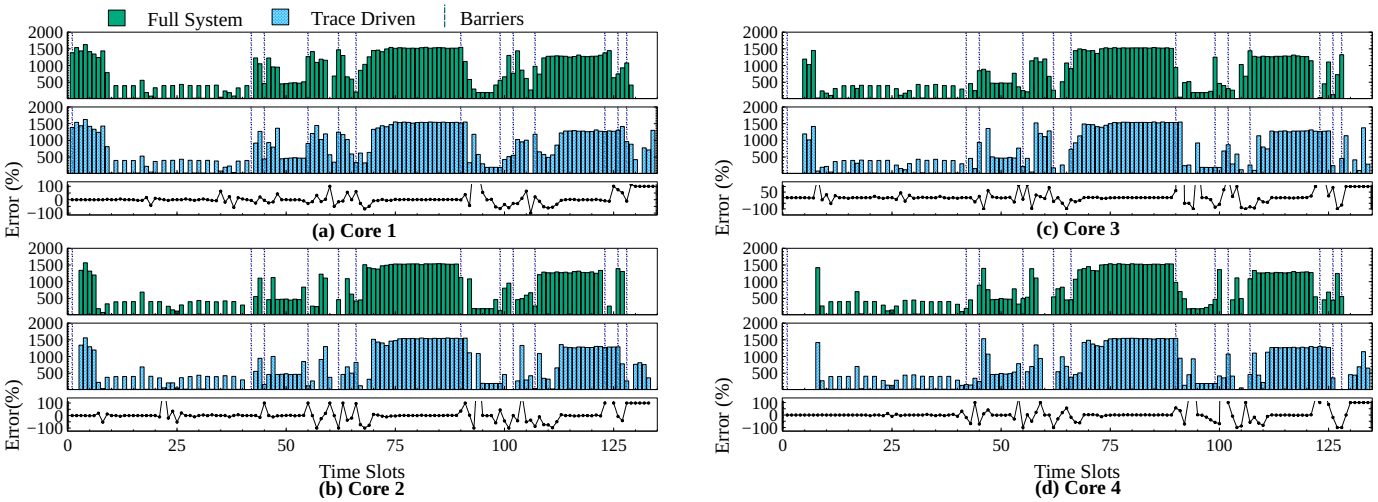

Fig. 6. Comparison of the numbers of cache misses: gem5 Full System vs. Trace-Driven simulation.

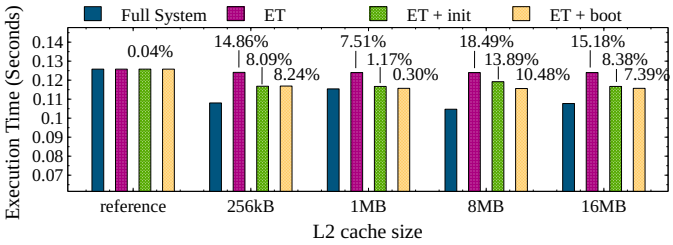

Fig. 7. Comparison of error for exploring the impact of L2 cache memory.

C. Consistency of Simulations in Presence of Dependencies

We evaluate the error percentage of the TD approach w.r.t. gem5 FS simulation in the case applications are composed of multiple threads with dependencies between them, e.g., induced by synchronizations. The mechanism that we defined for addressing this concern has been presented in Sections III-B. This is illustrated on the integer Radix sort kernel.

In the trace collection phase, the RP contains 4 cores and Radix kernel is executed with 4 threads. For each core, the numbers of cache misses observed during a given time slot and the induced error are illustrated in Fig.6. The moments at which synchronization barriers occur are highlighted by dashed vertical lines. We observe a fluctuation of the error on barriers occurrences. However, the observed peaks in the error are compensated during the entire simulation, so that the cumulated error is only 1.39%.

D. Trace Replication for Large-Scale Simulation

We evaluate the opportunity of simulating the replication of a given trace set on more processing elements than those used to collect this trace set. Typically, a trace obtained from a gem5 FS simulation on a 2-cores could be replicated on tens or hundreds of cores and simulated faster and still accurately with the proposed TD extension. A preliminary observation about cache miss behaviors of cores is that they are very similar (see Fig.6). The cache misses of all cores follow the same memory access time pattern. Then, such a pattern can be used as a possible trace template to be replicated among more cores. We explore different scenarios for identifying such a pattern by considering the MJPEG application, where traces are collected from: (i) 1-core RP, (ii) 2-cores RP, (iii) 4-cores RP and (iv) 8-cores RP.

The traces obtained from these scenarios are replicated on a platform with 8 TIs. We use normalized correlation function to produce an accurate estimation. Correlation is made between the number of cache misses obtained through the 8-cores gem5 FS simulation and the number of cache misses obtained via the above four scenarios. The obtained correlation coefficients for each scenario are shown in Table II. These results show that the three scenarios provide very similar behaviors regardless of the number of cores used. The produced execution time error comparing with the FS simulation is under 4%. Thus, the 2-cores scenario is relevant enough for a meaningful replication targeting up to hundreds of injectors. We used the above replication pattern to simulate a manycore architecture composed of up to 512 cores. The number of cores is limited by the host machine capabilities. The architectural exploration results are out of the scope of this paper.

In order to improve the correlation coefficients, we investigate the impact of application problem size on these coefficients. We focus on four applications: Radix, LU, Ocean and Barnes. For each of these applications, we chose three problem sizes, where Size 1 $<$ Size 2 $<$ Size 3. Here, size means the amount of processed data. Then, we collected their corresponding traces on the RP with four cores and four threads. Table III shows the average correlation coefficients calculated for each application trace according to the three problem sizes. Note the significant increase in correlation with larger problem

TABLE II
CORRELATATION OF TRACES FROM 1-,2-,4-,8-CORES RP COLLECTION.

Collection RP parameter	1-core	2-cores	4-cores	8-cores
Replication Ratio	8	4	2	1
Correlation Coefficient	0.77	0.80	0.76	0.99
Execution time error (%)	3.34	2.93	0.92	0.98

TABLE III
APPLICATION PROBLEM SIZE IMPACT ON CORRELATION COEFFICIENTS.

Application	Radix	LU	Ocean	Barnes
Problem size 1	0.82	0.65	0.57	0.59
Problem size 2	0.87	0.75	0.66	0.73
Problem size 3	0.90	0.90	0.80	0.80

sizes, originating from the increased pressure to memory subsystem: a higher dynamics in the cache miss rate over time results in more prominent correlation.

E. Trace-Driven Simulation Cost

We here analyse the breakdown of the simulation effort on the host machine for the following components: trace injector, cache, bus, memory and gem5 simulator. A single core Full System simulation is also given for reference. From the achieved experiments, we observed similar distributions that can be instantiated as in Fig.8 for the MJPEG decoder. The analysis is performed with the standard gprof profiling tool[1]. The most part of the TD simulation falls into the realization of cache coherent snooping protocol. Since the target system is bus-based, the memory traffic congestion and performance degradation is induced by increasing processors and injectors.

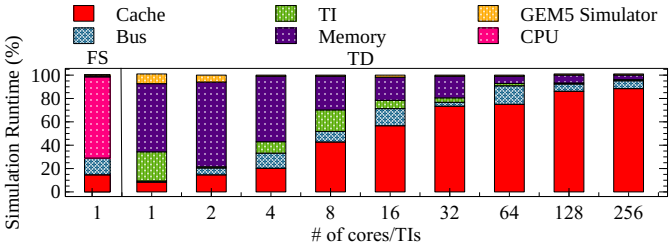

Fig. 8. Simulation time distribution among cache, bus, memory and gem5 simulator during trace-driven simulation.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we presented an extension of the well-known gem5 simulator with a trace-driven approach for an efficient exploration of manycore architectures. Our solution provides a fast and accurate simulation compared to the current gem5 version, while preserving all design capabilities of gem5. A number of experiments has been reported, showing all the benefits of our trace-driven extension: scalability via trace replication, speedup and accuracy. The obtained results showed a simulation speed of up to 800 times faster than the timing gem5 Full System mode, while the achieved accuracy varies from 0.02% to 6%. Our implementation aims to be made freely available online [2] for manycore architecture exploration, while decreasing the gap between simulation accuracy and performance.

Future work include improvements of the current trace-driven simulator by addressing issues concerning the optimization of trace file, the out-of-order processor support and different memory mapping algorithms. We also plan to support other programming models enabling the architecture exploration of complex computer systems like GPUs.

[1] http://sourceware.org/binutils/docs/gprof

[2] http://www.lirmm.fr/ADAC

VI. ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7/2007-2013) under the Mont-Blanc Project: http://www.montblanc-project.eu, grant agreement n° 288777.

REFERENCES

[1] J. Yi and D. Lilja, "Simulation of computer architectures: simulators, benchmarks, methodologies, and recommendations," *IEEE Transactions on Computers*, vol. 55, no. 3, pp. 268–280, 2006.

[2] "Gem5's memory system," http://www.m5sim.org, 2013.

[3] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, "Accuracy evaluation of gem5 simulator system," in *7th Int'l Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, 2012, pp. 1–7.

[4] "Open virtual platforms," http://www.ovpworld.org/, 2013.

[5] QEMU, "Qemu open source processor emulator," http://wiki.qemu.org/Main_Page, 2013.

[6] D. Sanchez and C. Kozyrakis, "Zsim: fast and accurate microarchitectural simulation of thousand-core systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 475–486. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485963

[7] S. Stattelmann, S. Ottlik, A. Viehl, O. Bringmann, and W. Rosenstiel, "Combining instruction set simulation and wcet analysis for embedded software performance estimation," in *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, 2012, pp. 295–298.

[8] J. Chen, M. Annavaram, and M. Dubois, "Slacksim: a platform for parallel simulations of cmps on cmps," *SIGARCH Comput. Archit. News*, vol. 37, no. 2, pp. 20–29, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1577129.1577134

[9] J. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, 2010, pp. 1–12.

[10] S. Abdi, G. Schirner, Y. Hwang, D. Gajski, and L. Yu, "Automatic tlm generation for early validation of multicore systems," *Design Test of Computers, IEEE*, vol. 28, no. 3, pp. 10–19, 2011.

[11] A. Rico, A. Duran, F. Cabarcas, Y. Etsion, A. Ramirez, and M. Valero, "Trace-driven simulation of multithreaded applications," in *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, 2011, pp. 87–96.

[12] S. Cho, S. Demetriades, S. Evans, L. Jin, H. Lee, K. Lee, and M. Moeng, "Tpts: A novel framework for very fast manycore processor architecture simulation," in *Int'l Conf. on Parallel Processing (ICPP)*, 2008, pp. 446–453.

[13] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[14] C. Minkenberg and G. Rodriguez, "Trace-driven co-simulation of high-performance computing systems using omnet++," in *2nd Int'l Conference on Simulation Tools and Techniques*, 2009, pp. 65:1–65:8.

[15] J. Hestness, B. Grot, and S. W. Keckler, "Netrace: Dependency-driven trace-based network-on-chip simulation," in *Proceedings of the Third International Workshop on Network on Chip Architectures*, ser. NoCArc '10. New York, NY, USA: ACM, 2010, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/1921249.1921258

[16] R. A. Uhlig, "Trap-driven Memory Simulation," Ph.D. dissertation, University of Michigan, 1995.

[17] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The splash-2 programs: characterization and methodological considerations," in *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*, 1995, pp. 24–36.