THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

High Speed Cycle-Approximate Simulation of Embedded Cache-Incoherent and Coherent Chip-Multiprocessors

OPEN ACCESS

CrossMark

High Speed Cycle-Approximate Simulation of Embedded Cache-Incoherent and Coherent Chip-Multiprocessors

Christopher Thompson[1] · **Miles Gould**[1] · **Nigel Topham**[1]

Abstract The increasing density of silicon processes, coupled with the development of ever more energy and space efficient embedded core designs, has led to multi-processor system-on-chip (MPSoC) designs becoming increasingly attractive for use in embedded systems. Unfortunately this increase in core count gives rise to an explosion in design space possibilities, especially when heterogeneous designs are considered. To address this problem, new techniques in simulation are required to increase the simulation performance of these systems, while maintaining the accuracy needed to make good design decisions, and to verify the performance characteristics for real-time systems. We present a new high-speed, near cycle-accurate simulator, addressing an important but neglected category of multicore systems: deeply-embedded cache-incoherent MPSoCs. We take advantage of the unique properties of these systems to relax synchronisation constraints and increase the parallelism of the simulation. In doing so we achieve performance not possible using previous simulation techniques, without compromising the accuracy of the results. Quantitative performance results are presented across a large range of simulated MPSoC designs, comprising 1–64 cores, on average we simulate at 5.7 MIPS, with simulation speeds reaching 377 MIPS in the best case. Comparing against FPGA implementations we demonstrate that the simulator manages this with an average timing error of only 2.1%. Applying some of

✉ Christopher Thompson
mm0zct@gmail.com

Miles Gould
miles@assyrian.org.uk

Nigel Topham
npt@staffmail.ed.ac.uk

1 Institute for Computing Systems Architecture, University of Edinburgh, Edinburgh,
United Kingdom

Springer

these techniques to coherent simulation enables even coherent 64-core designs to be simulated accurately at up to 2.2 MIPS.

1 Introduction

When designing a system on chip (SoC) for any system it is important to evaluate performance characteristics, but when designing for a high volume deeply embedded system it can be especially important to minimise the area (and as such cost) of the silicon needed for the chip, along with the power requirements. This usually means tuning the performance to only just meet the worst case performance requirements, and no more.

In order to find this optimal configuration many iterations of software development and system configurations are required, typically using slow cycle by cycle simulators, perhaps with the use of faster functional only simulators to aid software development. Unfortunately evaluating the performance of new software, and finding the most cost effective system, requires a slow cycle accurate simulation of every likely SoC configuration, and using a typical single threaded cycle accurate simulation does not allow fast turnaround for a programmer doing performance optimisations.

Hardware prototypes implemented in field programmable gate arrays (FPGAs) are a good platform for running and evaluating performance of software on a prospective hardware design, but require significant upfront synthesis time to generate a new design configuration. This means they are more useful for evaluating different software options, on a fixed hardware platform, rather than rapidly testing multiple hardware options.

The size constraints of FPGAs also mean that larger multi-processor SoC (MPSoC) designs are unlikely to be feasible, and must be tested in simulation. Expanding on these problems, FPGA development boards themselves are very expensive, so it is preferable for software developers to work in simulation, with the ubiquitous general-purpose computing power of compute clusters and server farms.

Many embedded systems use multiple processor cores without hardware cache-coherence. For example TI's automotive MPSoC family, Jacinto6 ECO also known as DRA72x, contains six mostly-incoherent processors: a single high-powered ARM A15, four low-power ARM M4 cores, and a single TI C66x DSP [1]. Providing hardware coherence adds an unnecessary hardware development expense, in implementation and verification, and more significantly in silicon area and system power consumption. It also adds complications when reasoning about worst-case execution time and performance for hard real-time systems. As transistor counts increase multicore embedded systems are becoming more common, and so cache-incoherent MPSoCs are becoming an increasingly important simulation target; yet recent advances in simulation technology have largely been applied to cache-coherent targets.

Traditional cycle-accurate simulators take a cycle-by-cycle approach to the process, modelling the pipeline of each core and interconnecting buses in a single thread. This is the easiest way to ensure timing-accurate functional behaviour, deterministic simulation, and correct evaluation of memory interleavings between cores. Functional-first simulators perform a high-speed functional evaluation of behaviour and then reconstruct timing data using a timing model. This approach yields greater simulation speed and can be easily parallelized, but it is difficult to extend accurately to the case of cache-coherent MPSoCs, in which timing influences the behaviour of cores. The problem has been tackled before [2] but this implementation requires an FPGA to process the timing model, and significant communication between the timing model and functional simulation. However, in the cache-incoherent case the timing interactions between cores are limited to cache misses, cache flushes and cache-bypassing memory operations, making accurate parallel functional-first simulation possible while maintaining high simulation speeds.

In this paper we present a novel approach to simulating these embedded systems, where decoupling the simulation of the cores and interconnect is exploited while still maintaining cycle-accurate timing behaviour. This results in accurate performance modelling at simulation speeds up to 378 MIPS, or 117 MHz, with an average speed of 5.7 MIPS and 10.1 MHz across a large design-space of over 22,000 design points. We also demonstrate that by applying many of these optimisations to a coherent simulation it is still possible to achieve state-of-the-art simulation speeds for manycore chip multi-processors (CMPs) without sacrificing accuracy.

The key contributions presented are:

– Leveraging incoherence to increase parallelism in the simulation.
– Efficient NoC simulation through packet tracking and cache friendly data structures.
– Significantly faster than state-of-the-art simulation without sacrificing accuracy for incoherent systems.
– Better than state-of-the-art simulation rates for coherent systems at the simulation accuracy provided.

1.1 A Note on Terminology

When discussing cycle-accurate and cycle-approximate simulators, care must be taken with the terms. In our discussion cycle-approximate refers to results which aim for cycle accuracy, but for fundamental reasons due to the technical approach cannot absolutely achieve it. Cycle-accurate should be used to refer to simulations which can produce true cycle-accurate results, or which are only incorrect due to engineering effort (e.g. pipeline edge cases that are not modelled due to human effort, rather than simulation cost). Unfortunately other papers claim cycle-accuracy when they are in fact cycle-approximate. Our simulator could be described as near-cycle-accurate, since we are more accurate than most "cycle-accurate" papers, and the only cache-incoherent circumstance where we violate cycle-accuracy is very rare, and could be mitigated with a change to the modelling of the core microarchitecture (we model instruction fetch too late, so functional data dependency can violate true cache-miss

timing). Modelling the instruction fetch more accurately, and ironing out any minor differences in the pipeline model relative to the RTL, would allow us to claim true cycle-accuracy for both incoherent and coherent simulation.

## 2 Cache-Incoherent Target Platform

The target platform for this multicore simulator is a custom NoC based MPSoC flow, using the 3-stage variant of the EnCore RISC microprocessor. The tool flow takes a high level MPSoC description and generates Verilog for FPGA and silicon implementation, and equivalent configuration files for the simulator. The platform comprises clusters of 1–8 processor cores, connected through a per-cluster arbiter to an ARM AMBA AXI [3] based packet switched network, which connects cores to memory banks and devices, such as the UART, real-time clock, and display controller. The design options are summarised in Table 1 with an overview diagram in Fig. 1. The complexity parameter influences the number of switches and number of layers in the switching network before adding cores and peripherals, extra switches are added if the configured complexity does not provide enough connectivity.

The interconnect comprises five independent channels, implemented as in the AXI standard: three from master to slave for sending read requests, write requests, and write data, and two return channels for read data and write-complete acknowledgement. By using dedicated channels for each category it is simple to avoid both deadlock and issues with bandwidth sharing between the different channels.

Table 1 MPSoC design configurations

Design parameter	Possible configurations
Core architecture	ARC700 32-bit RISC
Pipeline	3-stage in-order
D-Cache size	4 KB
D-Cache associativity	Direct mapped, 2-Way
I-Cache size	4 KB
I-Cache associativity	Direct mapped, 2-Way
Cache line size	32 Bytes
Interconnect protocol	AMBA AXI
Interconnect topology	32-bit wide omega network
Coherency protocol	None–Cache-Incoherent
Cores per cluster	1–8
Clusters	1–8
Block RAMs	1, 2, 4, 8
Total block RAM size	512 KB
Complexity	1, 2, 4, 8, 16
Fifo depth	2, 16
Core freq (MHz)	12.5, 25, 50
NoC freq (MHz)	12.5, 25, 50, 100

Fig. 1 Overview of the target architecture
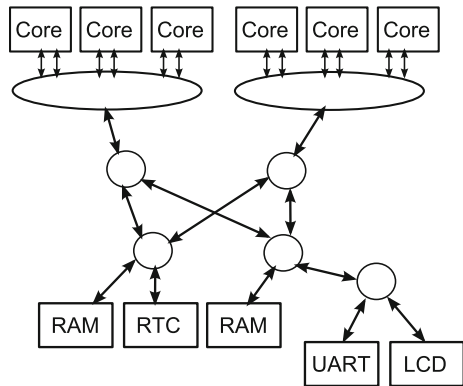
On the core, each cache has a unified output port, so read request, write request and write data must be time multiplexed out onto the AXI interface. The write request will issue before the associated data, but read requests are statically arbitrated with lower priority than write requests and above data, so are inserted in a cycle between write requests and the trailing write data. Since there is also an instruction cache with its own output port, it is statically arbitrated as higher priority than the data cache port, before being separated onto the discrete AXI channels.

The cluster level 8-way arbitration is implemented on a per channel basis (for the three outgoing channels) consisting of a three layer deep tree of two-way arbiters. Each arbiter contains a synchronous flip-flop to perform arbitration, which is toggled when the Ready line is asserted from the next arbiter, and a packet is ready to be sent from the previous. There are no other registers in the arbiters, just one final output register for the cluster, so the whole three level tree functions as a single cycle eight-way arbiter, with the correct arbiters being toggled due to the backpropagation of the ready signal through the arbiter network. The arbiters can also be configured to toggle arbitration only on "Last" packets, for multicycle data write-backs. With this enabled, data will arrive at the memory controller in a contiguous series of packets, rather than interleaved with data from other cores. Depending on the implementation of the memory controller, the performance difference can be significant, and a simple memory controller may not support interleaved data packets.

The rest of the switches in the system are two input two output, statically routed omega switches. The routes for different memory regions are stored in a small lookup table at the core/cluster level, and encoded into the AXI slave address signals, while the slave to master routing is performed by evaluating the route bit-string in reverse. Each output from the omega switch is essentially another two input arbiter, this time with the address bit check combined with the valid packet signal, and uses the same arbiter module as the cluster arbiter. Each switch can either be configured as registered or combinatorial, so portions of the network propagate in a single cycle.

Since there are five channels in the AXI standard implemented, even a simple two by two omega network (supporting up to four slaves, and four masters/clusters) requires 20 switches.

The cores are a modern ultra-low-power, silicon optimised three-stage RISC architecture, implementing the ARCompact instruction set. Implementation in a Xilinx Virtex-6 XC6VLX240T (a ML605 evaluation board) can be achieved at 75 MHz, with 65 nm LP implementation expected to hit 250 MHz. While it is possible to synthesize an FPGA design with 12 cores, as used for accuracy comparisons in Sect. 6, the University of Edinburgh has recently fabricated at chip at 65 nm with 32 cores in a similar configuration, for use in the transceiver of a new wireless network technology [4].

3 Motivation and Innovation

Simulation of this MPSoC was one of the driving factors for developing the simulator, to help with performance estimations and software development before the chip development completed. It is likely that there are many similar cases where a simulator such as this would be extremely beneficial.

It was realised when designing the simulator that the out of core traffic could be modelled much more efficiently than traditional coherent simulators. A timing accurate simulator for a coherent system must ensure that every single memory access happens in the correct order. This means that every memory access simulated must result in the simulation thread synchronising with a global clock reference, to ensure that all memory operations that should happen before it have taken place. This tight synchronisation causes problems with simulation performance, as each simulation thread can not do much work without synchronising; the Slacksim paper [5] discusses this well. With a cache incoherent system a correct program must already assume that cached memory operations will not be immediately available to another processor core, and cannot guarantee visibility until a cache flush operation. A well behaved program should also not assume that it is running on an incoherent system, as events may cause cache lines to be flushed before the programmer is expecting it, such as interrupts. This means that all cached memory accesses should be free of data races between cores, and these memory operations can be performed as soon as possible in the independent core threads. Now a significant synchronisation problem has been removed, and only modelling the timing side effects remains. To do this requires the communication of cache miss events to a separate thread to model the NoC interconnect, which takes care of the timing impact on the out of core traffic. For memory accesses which will cause data races or require synchronisation (i.e. cache bypassing operations and cache line flushes) the core thread is synchronised to the interconnect thread, and the timing model completes its operation before continuing core simulation. In doing so, timing correct behaviour is maintained, with significantly less synchronisation than traditional coherent simulators.

4 Simulator Implementation

The simulator used in the this paper is based on the ArcSim [6,7] simulator, developed at The University of Edinburgh. ArcSim is a high speed simulator designed to simulate the ARCompact RISC ISA for both functional simulation and cycle accurate simulation. ArcSim is compatible with the ARC600, ARC700 and ARCv2 ISAs, pro-

viding cycle accurate models for both 3, 5 and 7 stage interlocked in-order pipeline micro-architectures, with the 3 and 5 stage models developed around the EnCore microprocessor. ArcSim uses an asynchronous dynamic binary translator (DBT), or just in time compiler (JIT), to accelerate simulation by translating target instruction sequences into native code, while also supporting the inclusion of code to model the core micro-architectural state changes for the instruction sequence. This can result in simulation speeds of up to 1000 MIPS, or 100 MIPS in cycle accurate mode. Arc-Sim supports full system simulation, along with syscall emulation to enable user-land application simulation, or a hybrid simulation mode where a bare metal environment can run in full system simulation mode with IO devices and full control of the MMU, but system calls are still trapped to the host to allow file access and console IO support.

Multicore simulation has previously been developed in ArcSim, with the work presented in SAMOS 2011 [7] demonstrating functional simulation support for over 1024 cores at speeds in excess of 10,000 MIPS. The work presented here advances this previous work on ArcSim by implementing a decoupled interconnect model for multicore architectures.

When operating in cycle-accurate mode Arcsim reconstructs updates to the pipeline model after executing each target instruction (both in fully interpretive and JIT-compiled modes), and the core model simulation is explained in detail in the paper by Böhm et al. [6].

To parallelise the multicore simulation while providing cycle-accurate modelling of the shared NoC interconnect a similar approach to the FaCSim [8] and FAST [2,9,10] simulators is used, making use of lock-free asynchronous producer-consumer circular buffers to decouple simulation of the cores from the interconnect.

These relatively large buffers provide significant slack for the core simulation threads to execute ahead of the interconnect model, which means the interconnect model is rarely waiting for work to process, and helps to cover the time when the thread simulating a particular core is scheduled off by the host operating system. This increases the efficiency of the simulation when the host system has fewer physical cores than the target being simulated, and is enabled by exploiting the reduced synchronisation required for incoherent programs.

Unlike either FacSim or FAST, which decouple the functional simulation and the core timing model, Arcsim [6] models the core micro-architecture interleaved in the same thread as the functional core simulation. Each core is simulated in its own thread, which means that the core timing model is as parallelised as the functional simulation, helping maintain performance when larger multicore designs are simulated. Because the NoC architecture is cache incoherent, the core model can safely include the caches, meaning only cache misses and explicit cache-bypassing instructions must be communicated to the interconnect model. This reduction in communication allows the use of smaller communication buffers when compared to other decoupled simulators such as FAST, COTSon [11] and FaCSim, allowing larger target systems to be modelled on smaller hosts systems.

Figure 2 shows a simplified view of the memory components of the core simulation kernel, highlighting where asynchronous and synchronous communication is required, and the presence of both instruction and data caches in the core simulation.

```
simulate_instruction(){
  update_icache();
  ...
  if(memory_op)
    perform_memory_operation();
}

update_icache(){
  if(!i_cache->is_hit(pc)){
    interconnect_i_queue->push_fetch(pc);
  }
}

perform_memory_operation(){
  if(is_bypass){
    if(is_read){
      interconnect_d_queue.push_read(addr);
      while(!interconnect_d_queue.empty()){ os_yeild(); }
      r_data = interconnect->get_read_value(core_id);
    }else{
      interconnect_d_queue.push_write(addr, w_data);
      while(!interconnect_d_queue.empty()){ os_yeild(); }
    }
  }else{
    if(is_read){
      r_data = mem->read(addr);
    }else{
      m->write(addr, w_data);
    }
    if(!d_cache->is_hit(addr)){
      if(d_cache->is_dirty(addr){
        interconnect_d_queue.push_writeback(addr);
      }
      interconnect_d_queue.push_fetch(addr);
    }
  }
}
```

Fig. 2 Simplified processor simulation instruction implementation, demonstrating communication to the interconnect thread

Embedded cores, such as the ones used here, are often simple in-order interlocked pipelines, which stall on memory events such as cache misses. This allows the time passed for a processor core to be accurately described as the the sum of cycles spent internally, plus the cycles spent waiting for IO requests. This is the second property of the embedded cores leveraged to decouple the simulation to this extent, and is in some ways analogous to the QuantumKeeper in SystemC TLM2.0 [12]. Features such as victim caches and store buffers can still be accurately modelled so long as the interconnect model is aware of them. For example, a store buffer of depth N would mean that the interconnect can process up to N cache write-miss events in parallel before adding time to the core's IO-time offset, while a write-back buffer would be modelled in the interconnect itself.

The simulator is designed to make full use of modern multicore processors, similarly to FAST running parallel functional simulations. One host thread is used per simulated

Fig. 3 Overview of the incoherent simulation, threads are identified by dashed boundary boxes
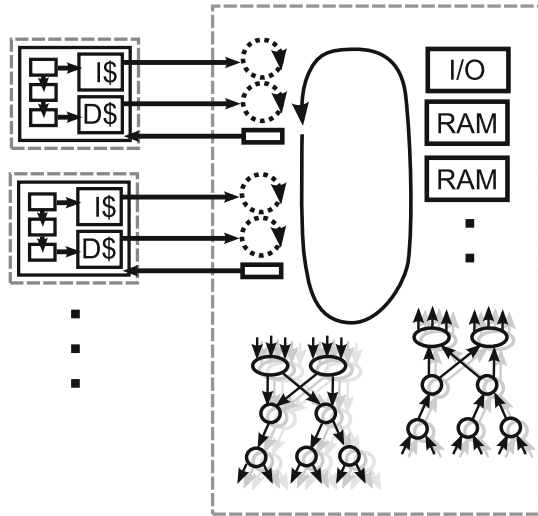
core, another thread for the interconnect, and additional threads for the functional simulation of IO devices such as display and sound devices. The simulator also supports parallel JIT worker threads to improve the performance of the core simulation. When running the interconnect thread, the simulator will use at least $N + 1$ host cores where N is the number of target cores being simulated. The core and interconnect simulation threads are shown in Fig. 3, with dashed box boundaries representing operating system threads and dashed circles representing asynchronous lock-free queues. Other threads such as virtual displays and JIT compilation threads are omitted for clarity.

4.1 Details of the NoC Interconnect

Based on the AXI protocol the NoC is implemented with five identically implemented channels, as described in Sect. 2. The design flow takes a list of 'master' nodes, 'slave' nodes, a design metric termed the 'complexity' of the network, and other configuration parameters. The complexity controls the width and depth of the omega network generated; then extra switches are added if nodes cannot be connected, and redundant switches are pruned. Omega networks use a multi-layer switch topology like the butterfly, comprising two input and two output switches. Unlike a traditional butterfly network, the presented network's source and destination nodes are not the same: there are explicit 'master' ports for cores and DMA-capable devices, and 'slave' ports for memory-mapped device and RAM interfaces. The tools then generate for the simulator an ordered list of these switches, with outputs and inputs explicitly connected with named 'wires', along with details of the master and slave connections and routing information based on memory addresses.

Additional complexity is introduced because not all switches are registered. They can optionally be configured with FIFO buffers, or they can act entirely combinatorially from one registered switch or input to the next registered switch or output in a single

cycle. In this mode they behave somewhat like an $N \times N$ crossbar, but with internal collisions. The simulator correctly handles these combinatorial switches while still modelling them in a modular per-switch fashion.

4.2 Modeling the NoC Interconnect

The interconnect model has two key roles in the simulation: not only must it model traffic and keep track of time spent in IO for each core, it must ensure timing-correct ordering of memory operations. Because the target platform is cache-incoherent, only cache-bypassing instructions must be committed in timing order to ensure timing-accurate behaviour. Because of this, cache transfers are modelled without actual data, and memory operations are committed by the core simulation thread. This is also how FaCSim operates, but because it only simulates a single core it does not need to worry about memory orderings. Cache miss operations are logged in the circular buffers by the core simulation and the core can continue simulation, but for cache-bypassing operations the core must wait until the buffer is flushed empty signalling the operation has been completed by the interconnect thread. This impacts performance somewhat; but it is required unless an alternative mechanism of ensuring correctness is employed in the functional simulation, such as checkpointing and roll-back as implemented in FAST [9].
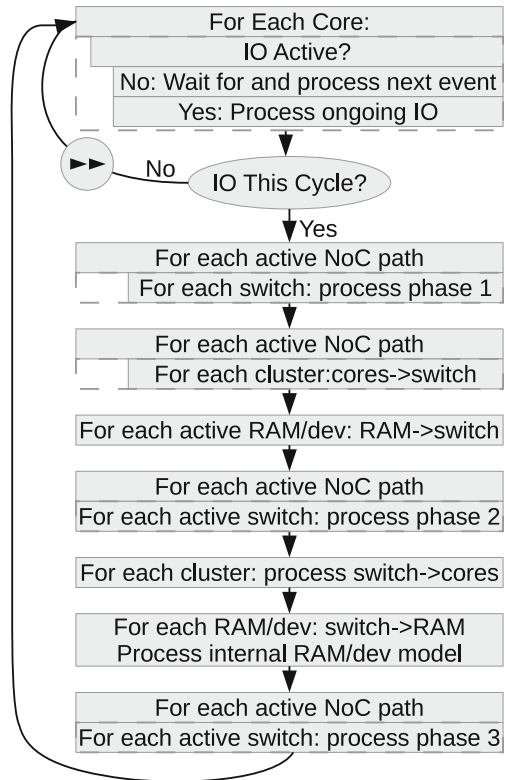
Rather than compute routing tables like the hardware, the simulator uses the routing tables from the NoC description to perform master to slave routing, and constructs the return routing bit-string dynamically as it passes through the network. Routing tables are shared between the switches on each channel for the same logical switch node.

Since Arcsim already provides extremely fast cycle-accurate simulation of the core micro-architecture, it was a challenge to model the interconnect and memory controllers at sufficient speed to avoid bottlenecking the whole system. To achieve this extensive use of packet counting and fast forwarding is used to ensure as little work as necessary is performed while processing the interconnect.

For example, each cycle the interconnect model checks to see if each core has an active transaction, and if not when the next transaction is due (by checking the pending transactions in the buffer). If there are no active transactions then the interconnect can directly skip ahead to the cycle of the next pending event. Also for each network channel packets are counted in and out, and only the switches on channels which are active are processed. A summary diagram of the main interconnect loop is provided in Fig. 4 demonstrating where various optimisations are performed, and how phases of computation are interleaved.

To compute a single cycle of the network the simulator iterates over all switches (in active channels) three times, marked as phases 1, 2 and 3 in the diagram. First all switches check their inputs for an incoming packet, flagging themselves as active if they take in data or still have data in a FIFO. Because the switches are correctly ordered in source→destination dependency order the combinatorial switches can simply propagate the packets here to their output: this is the same memory location as their destination's input, so they can be computed in a modular fashion rather than as a monolithic $N \times N$ switch. In the case where all switches are registered, packets can be

Fig. 4 Overview of the interconnect simulation loop identifying work-saving optimisations

| For Each Core: |
| IO Active? |
| No: Wait for and process next event |
| Yes: Process ongoing IO |

IO This Cycle? — No / Yes

| For each active NoC path |
| For each switch: process phase 1 |

| For each active NoC path |
| For each cluster:cores->switch |

| For each active RAM/dev: RAM->switch |

| For each active NoC path |
| For each active switch: process phase 2 |

| For each cluster: process switch->cores |

| For each RAM/dev: switch->RAM |
| Process internal RAM/dev model |

| For each active NoC path |
| For each active switch: process phase 3 |

consumed at this point; but when combinatorial switches are present they are left to be cleared up in phase three. The second phase is for registered switches to output their data packet onto the wire, depending on their internal state machine which is modelled as part of the switch micro-architecture. Finally the third phase is run in reverse direction, from destination to source, to track back and clear up the packets left when combinatorial switches are enabled. In this phase packet collisions are detected and the internal switch state machines and arbiters updated. The simulation also updates traffic counters on switches which successfully transferred a packet. The overhead of simulating combinatorial switches is a reduction in simulator performance of approximately 30% for a benchmark with moderate traffic, because of the extra book-keeping involved. It was disabled for this work, since all the designs tested used registered switches with FIFOs of depth 2 or 16 depending on the design. The output phase of connected devices is run between phase one and two, and the input phase between the second and third phases of the network.

Unlike FAST, this simulator does not have to model the data for all memory operations, only the information relevant to timing: the address, time, and size of cache misses. The exception is single transfer un-cached accesses which are processed in the micro-architecturally-accurate model of the memory controller or device model, ensuring the correct memory operation interleaving and behaviour on locked memory regions when atomic accesses are used.

4.3 Simulation Challenges

A potentially problematic feature of the functional-first simulation technique used in the simulator is that instruction-cache miss events are only realised after the completed execution of the previous instruction. This means that the event may only be discovered after the simulation of another memory event in a previous instruction, which it should have preceded. To solve this problem two buffers are implemented between the core and interconnect simulation components, one for each of the two pipeline stages which can generate memory events. When the interconnect reads off events from a core's buffers it waits until one of the following conditions is met before proceeding:

– There is an I-cache event, but no data events. It can continue because the data memory port cannot produce events which happen before this.
– There are entries in both buffers. It executes them according to their timestamps— potentially simultaneously in the same fashion as the core supports.
– There is a data entry and the core model has advanced beyond the point where an instruction cache event can over-take it, determined by the latency of the slowest instruction.
– There are no events but the core model has advanced beyond the point where any event could be generated for this cycle.
– There is a data event and the data buffer is flagged as containing a cache bypassing operation, which is preventing the core simulation from continuing. This is the only condition under which the core is allowed to violate the ordering of instruction and data memory events, because the simulation cannot continue without running the memory operation through the interconnect or potentially violating the ordering of data memory operations from different cores.

The violation in the final clause is an extremely rare event. The I-cache event is still modelled and its latency accounted for, just at slightly the wrong time. It is also possible to write code such that this never happens, if it is extremely important that the platform is simulated accurately.

Because the simulator's correctness relies on a correctly written target application, which does not have accidental cached read/write data sharing, a second mode of simulation is provided which moves the data cache model to the interconnect. This mode fully models the data in the caches, and the correct interleaving of data cache line reads and write backs in the memory controllers. It is almost completely functionally true to the target system, while still maintaining significant parallelism, but loses the performance benefits of the mode primarily discussed in this paper, falling back to performance of 1–2 MIPS for small scale MPSoCs. This mode is used for debugging target applications which exhibit signs of accidental data sharing, and for coherent simulation modes.

5 Performance Evaluation

Unfortunately there is no standard parallel benchmark suite for cache incoherent architectures, so various multiprogrammed workloads were composed using Coremark and a subset of the EEMBC [13] suite. The benchmarks used were restricted by the target

Table 2 Composition of multi-benchmark workloads

Workload	Contained benchmarks
1_0	imgdisp
1_1	fbital
1_2	conven
1_3	autcor
2_0	coremark, imgdisp
2_1	autcor, conven
2_2	fft, vinterb
2_3	viterb, imgdisp
3_0	conven(2), imgdisp
3_1	conven, coremark, cache_thrash
3_2	coremark, fbital, imgdisp
3_3	coremark, fbital(2)
4_0	conven(4)
4_1	autcor, conven, coremark(2)
4_2	autcor, fbital(2), imgdisp
4_3	autcor, fbital, fft, imgdisp
6_0	coremark(5), fft
6_1	autcor, fbital, viterb(2), cache_thrash(2)
6_2	conven, coremark(2), fbital, fft, viterb
6_3	autcor, conven, coremark, fft, imgdisp, cache_thrash

platform simulated, due to limited on-board memory and the capabilities of the current runtime only the following benchmarks could be run: Coremark, AutoCor, Conven, Fbital, FFT and Viterb. In addition to these standard benchmarks we also ran two memory bandwidth heavy in-house benchmarks: a panning image display benchmark 'imgdisp', and a synthetic cache-thrashing benchmark. The image display benchmark features both bad cache performance, due to its working set and access patterns, and extensive uncached IO to the display controller, which is treated in the simulation as a synchronising event, making it a good indicator of simulator performance for workloads which feature communication and are IO heavy. Between the EEMBC and Coremark benchmarks there are a wide range of runtime behaviours, and with in-house memory intensive benchmarks cover the spectrum of compute and memory intensive workloads, while still exposing enough code complexity to challenge the core micro-architectural model.

Statically scheduled parallel workloads for 1–64 tasks were composed by randomly selecting benchmarks from this set. The benchmarks used in the 1–6 task workloads also used for accuracy comparisons can be found in Table 2.

To give a clearer measure of how performance varies with the benchmark behaviour 1, 3, 6 and 12 thread single benchmark workloads were also composed for the EEMBC suite and Coremark benchmarks. These benchmarks were run on appropriately configured 1-, 3-, 6- and 12-core MPSoC designs and simulated on a dual socket, 6-core
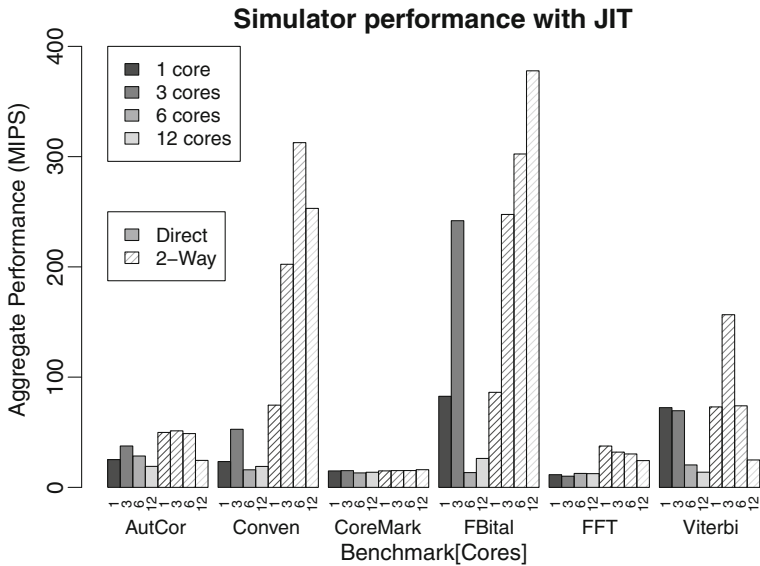
Fig. 5 Simulator performance across the EEMBC and Coremark benchmarks for 1-,3-,6- and 12-core designs. Solid bars indicate 4 KB direct-mapped simulated I and D caches, striped bars indicate 2-way set associative variants of the adjacent design

Intel Xeon X5650 workstation, providing 12 cores at 2.6 GHz. Performance results are shown in Fig. 5, with solid bars representing 4 KB direct-mapped caches, and striped bars representing 4 KB 2-way set associative caches.

The JIT worker threads were enabled for these experiments highlighting the best case for simulation performance. The 2-way set associative results for Conven and FBital demonstrate that extremely high simulation rates are possible for cache friendly benchmarks, with simulation rates over 377 MIPS, but the graphs also highlight that the simulator has two distinct interacting performance profiles. The core simulations run at approximately 14 MIPS per core in interpretive simulation, or 50–100 MIPS with the JIT enabled, and while the interconnect can keep up easily when there is minimal work to do, it can only simulate a saturated medium sized interconnect at around 1 MHz. Testing with a 64 thread cache thrashing workload on a 64-core target, resulting in 98.8% of each core's time being spent waiting for IO and only 0.006% of cycles able to be fast forwarded, gives a simulation rate of only 0.285 MHz on a 1.8 GHz 32-core server. However this still represents an aggregate simulation speed of 18.25 core-MHz, and despite the effective CPI of 170 provided a still competitive simulation rate of 0.107 MIPS.

By analysing the cache behaviour of the benchmarks in the simulations which produced Fig. 5 we can see the relationship between interconnect traffic and simulation performance. Figure 6 shows the breakdown of interconnect traffic across the simulations in terms of cache miss events per thousand instructions executed, isolating synchronising cache-bypassing traffic, data cache misses, and instruction cache misses. It can be seen comparing these two figures that good cache performance leads
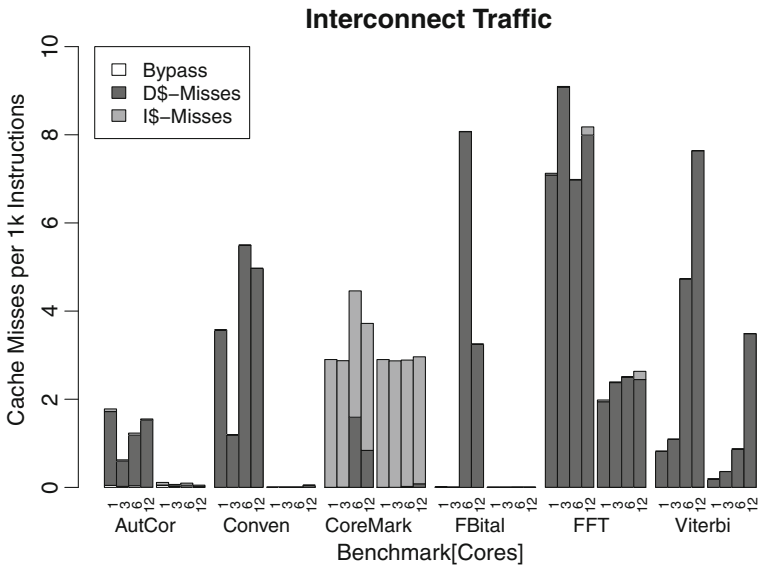
Fig. 6 Cache behaviour across the EEMBC and Coremark benchmarks for 1-,3-,6- and 12-core designs. Bars are arranged in the same order as Fig. 5, grouped into 4 KB direct-mapped and 2-way set associative caches

to exceptional simulation performance, although even when interconnect traffic causes the simulation performance to drop, it is still significantly faster than the state of the art.

Simulator performance statistics were also collected from a large scale design-space exploration on a shared cluster computing facility, comprising a mix of 8- and 12-core nodes. The simulations involved the previously discussed generated workloads, of which 64 different multiprogrammed workloads were produced comprising between 1 and 64 independent tasks. The designs were also randomly selected from a design space of 12,000 designs, varying the number of cores from 1 to 64, divided between 1 and 8 clusters. The other design parameters are listed in Table 1. In total 20,204 design-benchmark combinations were simulated, which used 1008 different MPSoC configurations from the total design space. For time-allocation on the compute cluster, the number of host cores requested for a specific design/benchmark configuration was the minimum of the number of cores in the design, and the number of tasks in the workload, plus one for the interconnect. This was chosen because cores will shut down once there are no tasks remaining to execute. The bare-metal runtime has an IO heavy start-up phase which must be executed on all cores of the design, so simulation speed can appear artificially limited by having to simulate many cores and an interconnect on only a few allocated physical cores during this time. The average simulation required 5.3 active cores, or 6 host threads.

For the large scale cluster experiments the results are presented in two parts. Firstly, Fig. 7a presents the performance in instruction throughput expressed in MIPS, as is typical for instruction set simulators. Here you can see the instruction throughput rarely
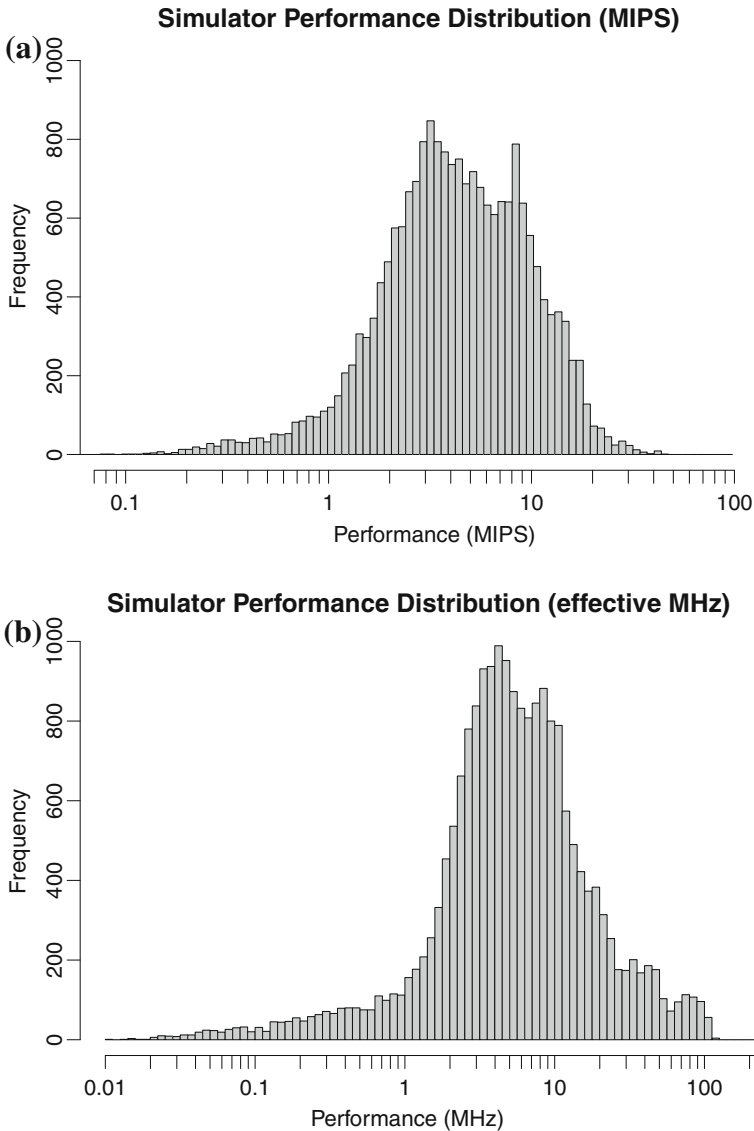
(a) Simulator Performance Distribution (MIPS)

(b) Simulator Performance Distribution (effective MHz)

Fig. 7 Histograms of simulator performance from a large scale design space exploration, showing that almost all of the 20,204 design/workload combinations have simulated performance over 1 MIPS, while most perform significantly better than this

drops below 1 MIPS (5%), and on average achieves 5.7 MIPS aggregate across the simulated cores. In the worst case there are almost no simulations which executed at under 0.3 MIPS (< 0.7%), and none below 0.08 MIPS, while the best-case simulations ran at an aggregate 45 MIPS.

Secondly, Fig. 7b presents the results as effective simulation rate of the whole MPSoC in MHz, not aggregated per core. For example a design with a 2:1 core

to interconnect clock ratio reported as 10 MHz, means that each core simulated at 10 MHz, and the interconnect at 5 MHz (since it only executes 1 cycle for every two the cores execute). Similarly if the core to interconnect ratio was 1:2, 10 MHz means that the cores simulated at 5 MHz each, while the interconnect ran at an effective 10 MHz. On average simulation rates of 10.1 MHz were achieved, only 7.5 times slower than the fastest design which can synthesize to a Virtex6 FPGA. The maximum performance was 117 MHz: significantly faster than the FPGA implementation, and approaching the 250 MHz projected speed for 65 nm silicon implementation. Unfortunately there is a very shallow tail below 1 MHz covering 8% of the simulations, which extends down to 0.01 MHz, indicating either pathological simulation conditions, or the simulation host in the cluster was being shared with an application causing a high degree of performance interference. The simulations were primarily run as a large design space simulation experiment, rather than for performance metrics, so time was not spent investigating the cause of the poor simulation performance of this tiny fraction of experiments. Looking at Fig. 8 however helps to explain some of the behaviour. With a maximum number of available host cores of 12, no simulations except a few with a single active core execute at less than 0.5 MIPS. All 1.8% of the designs which execute below 0.5 MIPS are running on over-loaded hosts, while on average simulation throughput was increasing with the number of target cores up to this point. This indicates that the performance was probably lost to the overhead of the operating system scheduler, and later work in this paper on a coherent simulation, employing a user-space scheduler, does not suffer from this problem. As such the worst results most likely could be redeemed through such a user-space scheduling system.

These figures for extremely stressed interconnect traffic do not present a realistic view of the simulator performance, since most workloads make effective use of the core's private caches to reduce IO traffic; the simulations used to generate Fig. 7 report an average IO/core cycles ratio of 7.4%, with a maximum of 93%. This is the reason for the extremely large performance distribution seen in Fig. 7a, which is unusual for a cycle-accurate simulator. The very lowest tails of the performance distribution are likely caused by simulations with more cores than host threads, which execute multiple instances of the panning image benchmark. Because this generates uncached IO to the screen, the core-interconnect synchronisation is triggered frequently. While not a problem in isolation, when the operating system cannot schedule all the threads to run simultaneously, it will often send the thread simulating the core performing IO to sleep, to let another core run. The interconnect thread will quickly process the single event and then hang waiting for the core to continue, because it cannot proceed beyond the current cycle time of the slowest core. With normal cache miss operations some of this scheduling overhead is masked by using large circular buffers, which allow the cores to execute sufficiently far ahead that the interconnect will not have to stall for long if the core thread is scheduled to sleep. This pathological behaviour is addressed later in Sect. 7 with the introduction of user-space scheduling, which provides significantly lower context switch overheads and allows for finer control of the scheduling. Other simulators also use user-space scheduling for the same reasons [14].

It is worth noting that while the simulator supports JIT compilation to accelerate simulation of the individual cores, it was disabled for the large-scale performance and accuracy results presented in Figs. 7 and 10. This is because there are a few cases
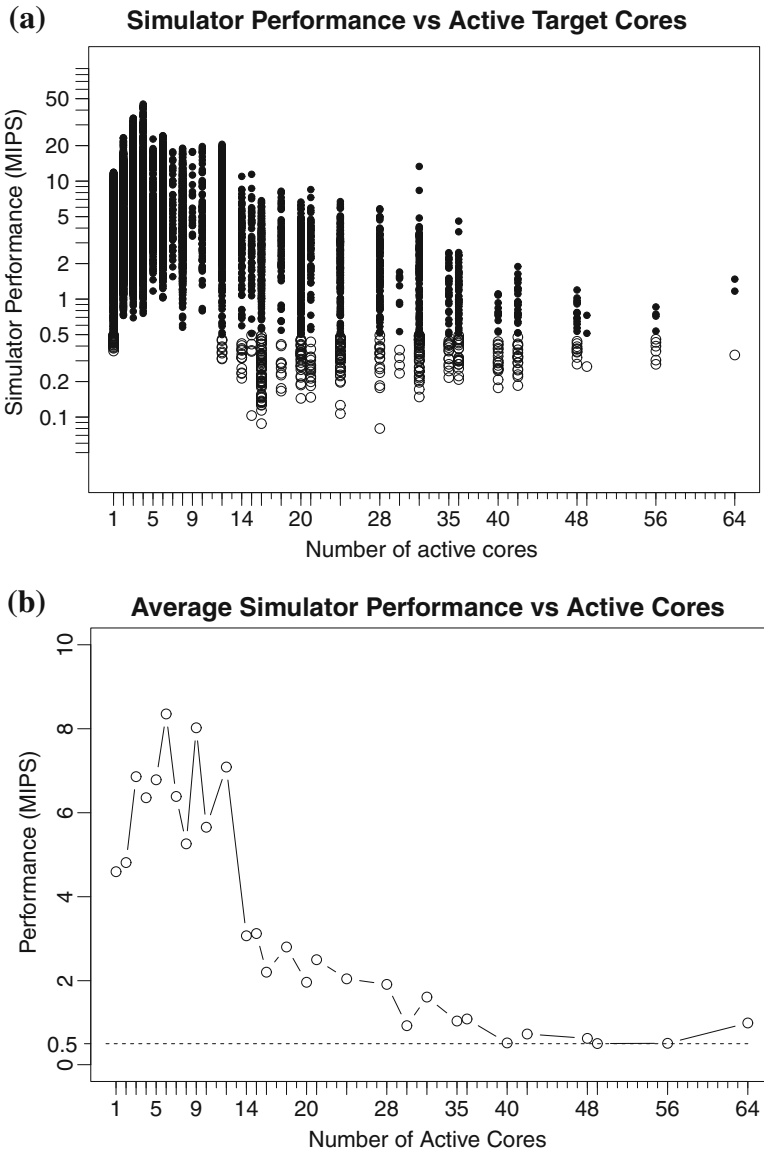
(a)   **Simulator Performance vs Active Target Cores**

(b)   **Average Simulator Performance vs Active Cores**

Fig. 8 Simulation performance against number of active simulated cores. This is one less than the required number of simulation threads; (a) shows the full distribution, with results less than 0.5 MIPS highlighted with hollow circles; (b) shows the average simulation performance, highlighting the impact of simulations with more threads than physical host cores

where the JIT compiled simulation model produces slightly different timing of events, due to a microarchitectural detail that has been updated in the interpreted mode only. This does not lead to a significant error in the simulated accuracy ($< 1\%$ difference to interpreted with the 3-stage pipeline), but does lead to non-determinism, as events

will be generated at slightly different times. Since the cluster-based simulations were run primarily for purposes of design-space exploration rather than performance evaluation of the simulator, it was decided that leaving the feature disabled was preferable. Since the extra performance provided by using the JIT has the most impact when the interconnect has little work and can fast-forward to keep up, its use would only stretch out the upper tail in simulation performance towards 100 MIPS per core. It would not significantly affect the body of the results, which are limited by the performance of the interconnect model.

6 Accuracy Evaluation

To construct and verify the detailed microarchitectural switch and memory controller models, detailed tracing output from the simulator was compared manually against Verilog simulation, confirming that the interconnect switch and memory controller implementations were indeed cycle accurate under a variety of test conditions. To empirically measure accuracy of the whole simulator several designs were synthesized to a Xilinx Virtex6 FPGA and test workloads were run on these platforms. The run-times in cycles for each core were collected and the results compared with the simulator, using the generated simulator configuration file.

To demonstrate how simulator accuracy varies with the different EEMBC benchmarks, and when scaling up the core count, the simulated cycle counts from the performance tests in Fig. 5 were compared with the same design running on the FPGA. Once again solid bars represent direct mapped caches, and striped are 2-way set associative, with each benchmark being run on 1-, 3-, 6- and 12-core designs. The bar for 12-core 2-way set associative is absent because this design does not fit into the available Virtex6 FPGA. The results, shown in Fig. 9 clearly demonstrate that the error is most affected by the benchmark, rather than the hardware configuration. Another feature which can be discerned using Figs. 5, 6 and 9 is that benchmarks with good cache behaviour (almost no interconnect traffic) such as FBital can have poor simulation accuracy, while those with poor simulation speed due to higher interconnect traffic like AutCor have very little error, indicating that the core microarchitectural model is more responsible for the error. The fact that error is most affected by benchmark and not size of the design supports this.

Secondly, as shown in Fig. 10, the randomly generated multiprogrammed workloads as described in Sect. 5 were run on a range of different MPSoC configurations, listed in Table 3, to evaluate error across different design options and more interesting workload combinations. Each shaded region represents the results for one workload, with each bar being the RMS error of the cores compared to the same core on the FPGA for a given MPSoC design. The designs are listed in order in Table 3. Here the trend that accuracy is determined by benchmark mostly continues, although more per-design variation is demonstrated than in Fig. 9 due to the shift in execution time spent in the interconnect and core respectively across the different designs.

The mean error from this larger experiment set is only 1.8%, with an RMS error of 2.1%, comparing well to the single core FaCSim, which achieves an average 7% error relative to its reference platform, and GEM5, which was recently evaluated to provide
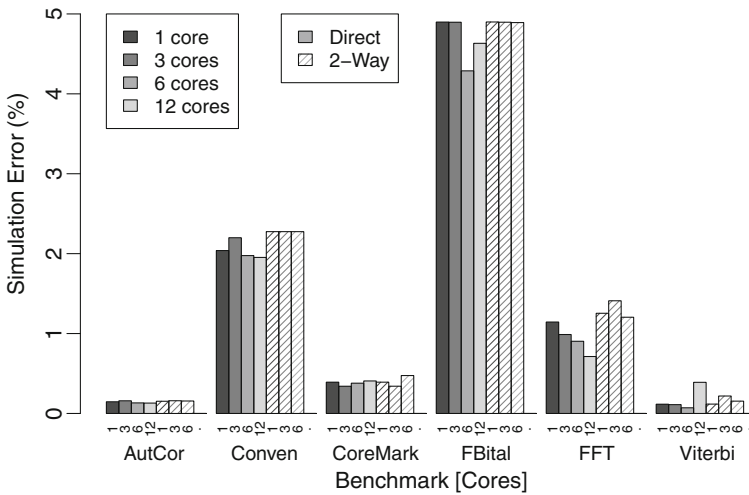
Fig. 9 Error from performance graph Fig. 5. Cycle count error from 1-,3-,6- and 12-core simulations of direct mapped and 2-way set associative configurations, for standard embedded benchmarks
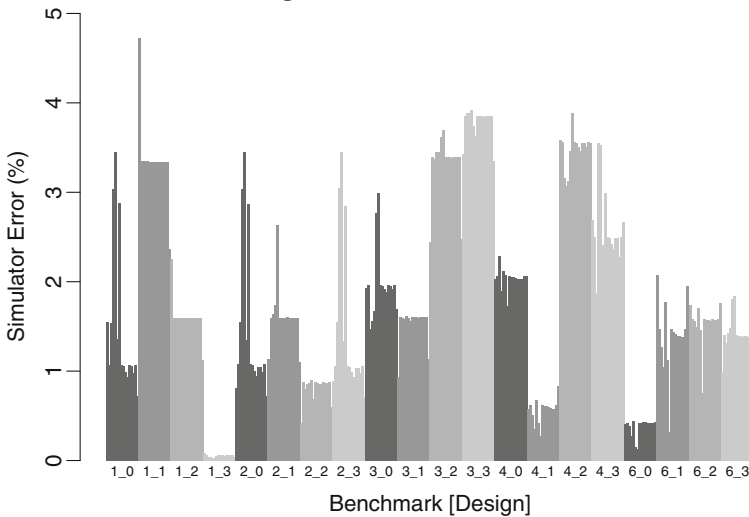

Fig. 10 Cycle count error across the small scale design space in Table 3 for mixed benchmark workloads detailed in Table 2

an average RMS error of 8.8% relative to a more complicated dual-core reference platform [15]. The most similar simulation system, a parallelised SystemC simulator, manages an average error of 6% [14] over a cycle accurate software model.

Table 3 NoC configurations used for accuracy analysis

Design	1	2	3	4	5	6	7	8
Cores	1	2	2	2	2	2	2	2
Clusters	1	1	1	1	1	1	1	1
RAMs	1	1	1	1	1	1	1	1
Complexity	2	2	2	2	2	2	2	2
Fifo depth	2	2	2	2	2	2	2	16
Core freq (MHz)	12.5	12.5	12.5	12.5	12.5	25	50	12.5
NoC freq (MHz)	12.5	12.5	25	50	100	12.5	12.5	12.5
Design	9	10	11	12	13	14	15	16
Cores	2	2	2	2	2	2	2	4
Clusters	1	1	1	1	1	1	2	2
RAMs	1	1	1	2	4	8	1	1
Complexity	4	8	16	2	2	2	2	2
Fifo depth	2	2	2	2	2	2	2	2
Core freq (MHz)	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5
NoC freq (MHz)	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5

7 Cache-Coherent Simulation

7.1 Target Platform

Based on the same 3-stage core, we proposed a manycore architecture for coherency protocol research. This is based on a more regularly structured mesh-tree hybrid architecture, with 16 nodes arranged in a 4 × 4 mesh. Each node contains an L2 cache, a narrower binary tree network with bandwidth adapter, and a power of two number of processor cores. Rather than the physically isolated channels of the AXI network, this architecture uses a unified-bus packet based protocol, in order to reduce the number of switches and wires required for the large scale design. Coherency is provided through a centralised directory with a separate binary tree network. This network provides two channels from the cores to the directory, requests and responses, and two channels from the directory to the cores, a unicast request channel for messages which can require multiple cycles to process or require a response (Exclusive invalidations), and a multicast channel for messages which can be processed in a single cycle and do not require a response. This means that the multicast network can never stall, reducing the requirements for buffering and back-channel signalling. This is possible because the coherency protocol uses silent invalidations for Shared state cache lines. The architecture is summarised on Table 4.

Because there is no physical implementation of this architecture, its purpose is for more abstract design exploration, the switches and routers have a simpler micro-architectural model to the previous cache-incoherent platform. However, the mesh network does consist of significantly more complex 5 × 5 routers with two arbitrated virtual channels, and all network data is transferred through the network to provide

Table 4 Manycore design configurations

| Design parameter	Possible configurations
Core architecture	ARC700 32-bit RISC
Pipeline	3-stage in-order
D-Cache configuration	4 KB, direct mapped
I-Cache configuration	4 KB, direct mapped
Cache line size	64 Bytes
Data interconnect protocol	Unified channel—packet based
Data interconnect topology	256-bit wide 4×4 mesh with two virtual channels
	64-bit wide binary-trees from each node, no virtual channels
	Both have one physical channel per direction
Coherency protocol	MESI directory protocol with coarse vector sharer encoding
Coherency interconnect	Single binary tree
	Two physical channels per direction
Nodes	16
Cores per node	1, 2, 4, 8, 16, 32, 64
L2 Caches	One per node, perfect caches
Core freq	800 MHz
NoC freq	800 MHz

full wire-level energy modelling. As such, while the switch models are slightly less detailed, there is at least as much computational effort required to simulate the interconnect.

The coherent architecture and its simulator are discussed in greater depth in later work [16], along with techniques for managing scalable cache-coherency, and simulation and microarchitectural efficiency of synchronisation primitives.

7.2 Simulation Details

The greatest challenge to simulating coherent systems comes when there are more target cores than host cores, and the host processor cores must be time-multiplexed to process multiple target cores. Unlike with cache-incoherent systems, each memory access must be synchronising for cycle accurate simulation, so buffering up events in large queues cannot be used to mask the overhead of operating system context switching. Figures 8a, b shows the performance impact of involving the operating system scheduler to switch simulation threads, incurring a significant penalty even with the large asynchronous buffers used to help mask the overhead. To address this shortcoming we implemented user-space scheduling, with cooperative multithreading, to perform low overhead context-switching between the simulated cores. Because the core simulation must now communicate for all memory accesses, not just cache misses, and all accesses are synchronising, the frequency of context switches is much higher than with the incoherent simulation, fortunately the user-space approach provides

Fig. 11 Overview of the coherent simulation, threads are identified by dashed boundary boxes
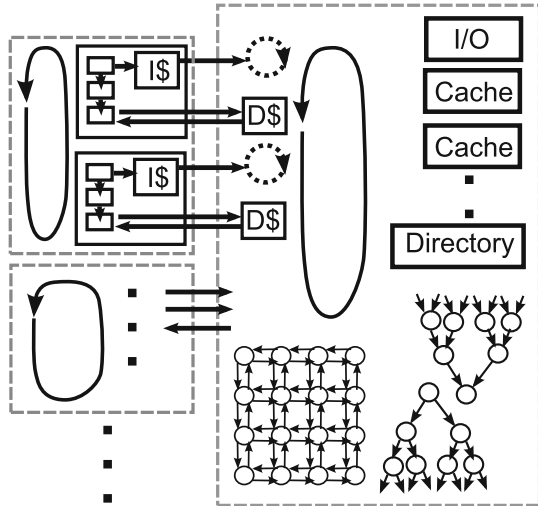

```
simulate_instruction (){
  update_icache ();
  ...
  if (memory_op)
    perform_memory_operation ();
}

update_icache (){
  if (! i_cache->is_hit (pc)){
    interconnect_i_queue->push_fetch (pc);
  }
}

perform_memory_operation (){
    if (is_read){
      interconnect_d_queue.push_read (addr);
      while (! interconnect_d_queue.empty ()){ coop_yeild (); }
      r_data = interconnect->get_read_value (core_id);
    } else {
      interconnect_d_queue.push_write (addr, w_data);
      while (! interconnect_d_queue.empty ()){ coop_yeild (); }
    }
  }
}
```

Fig. 12 Simplified processor simulation instruction implementation, demonstrating coherent communication to the interconnect thread

sufficient performance. This change in simulator organisation can be seen in Fig. 11, with the corresponding core simulation pseudo-code in Fig. 12, contrasting against the original Figs. 2 and 3. The user-space scheduler is a simple round-robin scheduler with the number of target cores divided equally between the allocated physical cores. The simulations used in this paper use four threads for processing core simulations,
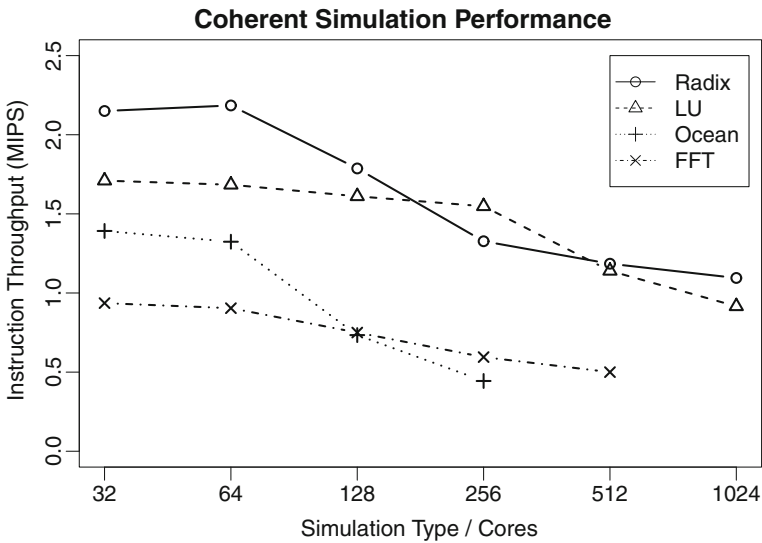
Coherent Simulation Performance

Fig. 13 Simulation performance of a large scale coherent manycore

Table 5 Benchmark configurations

Benchmark	Configuration
Radix	1,048,576 Keys
LU (contiguous)	512×512 Matrix
Ocean (contiguous)	258×258 Ocean
FFT	2^{20} points

and one for interconnect simulation (although the interconnect simulation supports a degree of parallelism, this was not used for these results).

7.3 Coherent Performance

Figure 13 shows the simulation throughput of this new coherent simulator while executing the Splash2 [17] benchmarks listed in Table 5, while utilising five operating system threads on the same 12-core workstation as used earlier. Despite the significant increase in work required by the interconnect thread, and the significant increase in the number of target cores simulated per host core (up to 256 cores per host processor for the 1024-core instances) the simulation shows remarkable performance and performance stability compared to using operating system scheduling, as seen in Fig. 8. By combining the optimisation to interconnect simulation and core-interconnect synchronisation, with high-performance user-space multithreading, it becomes possible to simulate massively parallel systems at much greater rates than other cycle accurate simulators such as GEM5, achieving the performance of Sniper without the accuracy sacrifices nor the use of binary instrumented execution.

8 Related Work

This section discusses some of the more relevant simulators and how they relate to the work presented in this paper.

Many of the simulators used commercially and academically for cycle accurate or approximate simulation are designed around single threaded discrete event simulation, with SystemC based simulation, Simics, and Gem5 based simulations being the most popular.

Because many of these are based on a functional (untimed) simulator we will summarise them first. However the lack of timing synchronisation or performance modelling, makes them unsuitable for micro-architectural research or design space exploration when used standalone.

8.1 Functional Simulators

8.1.1 Simics

Simics [18] is an emulation based functional full system simulator supporting multiple ISAs, now owned by Intel. It has been used to provide the core functional simulation to a number of cycle accurate simulators, and competes with TLM based SystemC models in industry.

8.1.2 Parallel Embra

Parallel Embra [19], like ArcSim, is a fast functional simulator for shared-memory multiprocessors, part of the Parallel SimOS complete machine simulator [20]. While Parallel Embra shares its use of binary translation with ArcSim it lacks its scalability and parallel JIT translation facility.

8.1.3 Mambo & MalSim

Another effort to parallelise a complete machine software simulator was undertaken with Mambo [21]. It aims to produce a fast functional simulator by extending a binary translation based emulation mode; published results include a speedup of up to 3.8 for a 4-way parallel simulation. Similarly, the MalSim [22] parallel functional simulator has only been evaluated for workloads comprising up to 16 threads.

8.2 Cycle Accurate Simulators

8.2.1 GEM5

GEM5 [23] is currently one of the most popular tools for single and multi-processor system simulation, and design-space exploration. Based on the Gems [24] simulator, but substituting Simics for M5 as the functional core simulator, the tool is fully open source, making it especially attractive for academic research. It uses emulation based

simulation to provide simulation models for multiple ISAs, while also supporting a variety of micro-architectural models for each core type. The interconnect is also modelled with a similar degree of configurability, with the most detailed interconnect simulations provided by the Ruby Garnet models, while simpler and faster models are supported with the Ruby Simple interconnect models and GEM5 Classic shared bus based models. GEM5's accuracy has been evaluated against a dual-core reference platform [15] demonstrating up to 35% error, but across Splash2 and ALPBench it achieved an RMS error of 8.8%. Unfortunately GEM5 suffers from relatively poor simulation speed. Running on the same compute cluster as used in this work, the ECDF [25], it provides simulation speeds of 0.06 MIPS to 0.275 MIPS with an average speed of 0.157 MIPS when simulating 4- to 16-core shared memory systems with the simple Atomic core model and Ruby interconnect model. Its timing/control-flow only based modelling means that effective wire level power modelling cannot be performed accurately, and relaxed memory consistency models cannot be accurately modelled or verified because the absence of data caching means that host memory consistency and coherency is applied, regardless of that specified in the model's coherency protocol.

8.2.2 ARMn

ARMn is a simulator designed to simulate multicore ARM systems [26], which connects multiple cycle accurate processor elements based on Simit-ARM together with a SystemC interconnect model. Out of the box it provides a simulation infrastructure with a configurable interconnect model with full cycle accurate simulation. However it only does so for a message passing API and does not model a shared memory system, requiring programs to be written against a special message passing library. The simulations speeds achieved are also not particularly fast, with reported speeds of under 7K cycles/s for a 16 node torus coupled to a synthetic traffic generator, although a simple four node bus can be simulated at a little over 400K cycles/s.

8.2.3 SimpleScalar

SimpleScalar [27] is another historically popular simulator, capable of functional down to detailed cycle accurate simulation. It has been parallelised by Zhong et al. [28], but SimpleScalar must run its own ISA, requiring its own compiler, and does not offer the simulation performance of more modern simulators.

8.3 Loosely Timed and Flexible Simulators

8.3.1 SystemC

One of the biggest industry tools for device, processor, and full platform simulation is the SystemC [29] modelling language, based around heavily templated C++ with a main simulation kernel handling the event loop, and extensive signalling and transaction support. SystemC can be used to write cycle by cycle models, through loosely timed transactional models, to purely functional simulation. SystemC is a powerful

simulation tool because of its flexibility and the large library of models already available, but its performance in cycle accurate and even functional simulation is poor compared to other dedicated simulators, partly due to the fact it runs in a single thread, but also because of the modelling abstraction overheads..

8.3.2 SimFlex

SimFlex [30] uses Simics as its functional emulation core, but uses statistical sampling of the application to avoid the overhead of simulating each instruction one by one with a timing model. This enables good performance, but accuracy suffers because it does not observe the entire application behaviour.

8.4 Parallel Relaxed System Simulators

8.4.1 BigSim

BigSim [31] is a modern multiprocessor simulator designed to deliver scalability and performance estimates on current and future large scale super computers. Message passing based applications developed with MPI or Charm++ are run on a system much smaller than the simulation target, but with as many threads as would be run on the real target, capturing the messages passed through the API with timing information. The timing estimations are made for code sequences but BigSim does not currently support a cycle accurate simulation of the target platform processors. Timing estimates for the interconnect are based on the latency of a message through the network topology of the simulation target, under infinite bandwidth/zero congestion circumstances.

8.4.2 FastMP

FastMP [32] attempts to address the issue of simulation scalability for multicore platforms. Their platform uses checkpoint based sampled execution to minimise the simulation work required, and analyses the discrepancy between the CPI of each core and the average CPI across all simulated cores to try and address errors in simulation accuracy at runtime. Unfortunately FastMP can only simulate applications which do not share data between threads.

8.4.3 Wisconsin Wind Tunnel

The Wisconsin Wind Tunnel (WWT) [33] is one of the earliest parallel simulators, but unfortunately requires applications to use an explicit interface for shared memory. Its direct execution simulation method also limits it to running on CM-5 machines, making it impractical for modern use. WWT II is the evolution of the first generation WWT, and in Mukherjee et al. [34] transition the WWT II methodology to other host architectures. Unfortunately it still does not model anything other than the original target memory system, and requires applications to be modified to explicitly allocate shared memory blocks.

8.4.4 Hornet

Hornet [35] is a scalable cycle-accurate simulator for on chip interconnects. The traffic can be fed from simulation (such as the MIPS simulator that is integrated into HORNET), from previously generated traces, or from instrumented natively executing code, such as with Pin [36]. HORNET is also integrated with a power model based on ORION 2.0 [37] and thermal model using HotSpot 5.0 [38]. This simulator is demonstrated to show good scalability up to 24 host cores, for 64- and 1024-core mesh architectures, but unfortunately does not give any absolute performance figures for comparison. The paper confirms that for accurate results the simulation feeding the interconnect model must be run with the interconnect simulation providing feedback, otherwise the interconnect injection rates could be much higher (since the cores do not wait the correct delay before the next request, as they assume an ideal interconnect), resulting in lower execution time. The paper also indicates that congestion modelling is only significant in bandwidth heavy applications, with benchmarks that do not stress the interconnect showing minimal error when congestion is not modelled. Benchmarks that do produce a lot of interconnect traffic exhibit significant error if the congestion is not modelled however.

Hornet does support a cycle-by-cycle execution mode, but has not been verified against a target platform, which means it is only useful for reasoning about abstract design-space exploration.

8.4.5 Marss

Marss [39] is a "cycle-accurate" multicore $\times 86$ simulator based on QEMU [40] and PTLsim [39], designed to model modern superscalar $\times 86$ architectures. It supports relatively complex interconnect architectures, but is limited to $\times 86$ simulation, and is also non deterministic. Being limited to $\times 86$ means that it is not suitable for experimentation with modifications to the ISA, such as adding new instructions. It is also not one of the ISAs used by low power deeply embedded cores likely to be found in a specialised manycore processor design, although it is used in the Xeon-Phi accelerators [41]. By being non-deterministic Marss demonstrates that it is not performing a true cycle-accurate simulation, and some event timings and relative orderings are being decided by host execution order, rather than strict timing model order. This makes performing experiments on features such as relaxed memory consistency models difficult, if not impossible.

8.4.6 SlackSim

Slacksim [5] is one of the early simulators exploring the relationship between synchronisation granularity and accuracy. It is a parallelised cycle by cycle simulator which simulates cores with caches in different threads, and allows for configurable synchronisation slack between the components. The simplest form is to synchronise with a barrier every N cycles, with synchronisation every cycle providing full cycle accuracy, and increasing error as the synchronisation period, or quantum, is increased. SlackSim also introduces a different form of relaxation, where the difference in cycle

time between the slowest thread and fastest thread is maintained within a defined slack. Similarly to relaxing the simulation period, relaxing the slack allows one to trade off performance for accuracy. Like many of the simulators discussed, the accuracy of SlackSim has never been verified against existing hardware. The performance of SlackSim is typically around 100 KIPS, while simulating a 4-way out-of-order processor without a detailed interconnect simulation.

8.4.7 FaCSim

FacSim [8] is a single core simulator which decouples functional simulation from micro-architectural and memory hierarchy simulation. Because there is only a single simulated processor there is no need to simulate memory contention or coherence traffic, and no potential for violation of memory orderings. This allows FaCSim to run an unbounded, fast, functional only, simulation to generate the program instruction stream and then feed it asynchronously to an optimised timing model of the processor and memory hierarchy. The models are connected through a shared memory circular buffer, making efficient use of a dual core host, and stall the functional simulation when the buffer becomes full. The single core simulation also means that the interconnect timing model can be extremely simple to calculate, as there is no opportunity for contention between multiple memory requests, which would require a more detailed simulation that a simple latency calculation based upon link distance and bandwidth. FacSim provides simulation rates up to 4 MIPS and has been shown to produce only a 6.8% root-mean-squared (RMS) timing error relative to reference hardware. This is not as fast as modern just in time compiled (JIT) simulators which compile the core timing model into the translated functional simulation, but is a good demonstration of the strengths of decoupling simulation components to improve performance through increased parallelism.

8.4.8 Graphite

Graphite [42] is a distributed multicore simulator that uses Pin [36] to instrument a natively executable application. The interconnect and distributed shared caches are simulated in parallel and can also be distributed across multiple nodes along with the functional execution. Graphite allows the different timing and functional components of the simulation to run asynchronously within a configurable bounded slack. The three supplied methods of synchronisation are a lax barrier, which keeps all simulations components running close to lock-step, synchronising every N cycles, lax peer-to-peer synchronisation, which puts cores which are too far ahead of the slowest core to sleep briefly, similar to SlackSim, and lax synchronisation. The last method uses timestamped messages from the different components to estimate the global clock locally, and never suspends simulation components for the sake of timing synchronisation. In all methods events are processed in the order they are received, and not reordered into correct timing order. As a result, although Graphite provides a scalable reasonably performant multicore simulation infrastructure, the accuracy is not high enough for some more subtle micro-architecture experiments. For example an interconnect which arbitrates on a per-flit basis rather than a per-packet basis will result

in interleaved flits from multiple packets, depending on the design of the memory controller this could be significantly worse, or better, for performance. By processing events in arrival order rather than timing order, this detail would be lost to a Graphite simulation.

8.4.9 Sniper

Sniper [43] is based on Graphite, with the Pin based functional simulation replaced by an interval simulation technique. The authors claim that this should provide greater accuracy for complex architectures than the in-order model which Graphite models. Unfortunately their accuracy evaluation against a real world Intel Core-2 based system resulted in, on average, 25% error. This poor accuracy provides strong evidence that the cycle approximate techniques used by many of these multi-processor simulators is insufficiently detailed for micro-architectural experimentation. It is also unsuitable for performance profiling of systems which are extremely timing sensitive, such as hard-realtime systems.

Sniper claims up to 2 MIPS performance when simulating a 16-core target on an 8-core host, approximately twice that of Graphite. In part this may be due to differences in interconnect complexity modelled, and performance of the host machine. Although simulating quite different architectures, our presented simulator manages over 2 MIPS simulating a coherent 64-core target using only 5 host cores, without sacrificing cycle accuracy.

8.4.10 Zsim

Another of the instrumentation based $\times 86$ multicore simulators, Zsim at first glance appears to be the holy grail of large scale manycore system simulation [44]. The paper presents a simulator apparently capable of simulating up to 1024 cores, with performance up to 1123 MIPS using a simple core model with interconnect contention, while demonstrating that relative to a 6-core system it can on average provide performance accuracy of 10% error. It achieves this by running a period simulation between 1 and 10K cycles where core models are processed for all simulated cores, generating memory events into a shared memory data structure, then running a model of the memory hierarchy to compute the timing effects, before returning for another period of core simulation. If your end goal is simply to measure final figure performance on a given benchmark and simulated system, then this might well be sufficient, and certainly outperforms the other $\times 86$ instrumentation based simulators such as Sniper. However unlike full cycle accurate simulators it does not accurately simulate functional memory ordering within the 1–10K cycle simulation phases, and it assumes that core-to-core interference events (such as coherence evictions) are rare. This assumption breaks down with low associativity shared resources, such as lower cache levels or directories, meaning the simulation will not be accurate to a designer trying to test the limits of their directory associativity or sizing options. Non cycle-accurate memory orderings can also severely distort spin-wait style timing statistics, such as the average time spent waiting on a spin-lock, or at a barrier.

8.4.11 HP COTSon

HP's COTSon simulator [45] uses AMD's SimNow™ for functional modeling and suffers from some of the same problems as SimFlex [30] and Gems [24].

Monchiero et al. have presented a methodology to simulate shared-memory multiprocessors composed of hundreds of cores [11]. The basic idea is to use thread-level parallelism in the software system and translate it into core-level parallelism in the simulated world. The existing COTSon simulator is first augmented to identify and separate the instruction streams belonging to the different software threads. Then, the simulator dynamically maps each instruction flow to the corresponding core of the target multi-core architecture, taking into account the inherent thread synchronisation of the running applications. This approach treats the functional simulator as a monolithic block, thus requiring an intermediate step for de-interleaving instructions belonging to different application threads. Monchiero reports this simulator performs at 1 MIPS for a single core simulation, scaling down to 0.7 MIPS for 1024 cores.

8.5 FPGA Accelerated Simulation

The two main limitations to FPGA accelerated platforms such as FAST, RAMP Gold, and ProtoFlex, are the cost of an FPGA platform to perform the simulation, and the limit to the number of simulatable cores imposed by the limited resources of the FPGA. This is unlike software simulation, where adding more cores may reduce performance, but the amount of RAM required to simulate even 1024 cores is well within the constraints of typical consumer hardware.

8.5.1 ProtoFlex

ProtoFlex [46] achieves reasonably high speed functional multicore simulation performing most of the simulation in FPGAs. Targeting the SPARC architecture ProtoFlex uses a pipelined core simulation engine called BlueSPARC in FPGA to simulate up to 16 instances of a SPARC processor model,with the aim to scale up in future by adding more engines. ProtoFlex supports full system simulation by falling back to a Simics based simulation when the FPGA model cannot simulate some part of the simulation. This reduction in simulation efficiency can be reduced by using a on-FPGA embedded or soft processor to run the software fall-back model rather than require the high-latency communication to the host computer. The 16-core simulation achieves simulation rates up to 62 MIPS, and even the theoretical 100 MIPS throughput of the BlueSPARC engine is significantly slower than modern JIT compiled software simulators (typically several hundred MIPS per core, with results up to 1323 MIPS per core achieved by the current state of the art [47]).

Being already in-FPGA allows for the memory trace to be easily fed into FACS FPGA cache model, which is capable of modelling the CMP cache model at full speed, except for very memory active periods of simulation, and this allows for high speed "functional warming" of the cache state for a statistical sampling based cycle accurate model.

8.5.2 RAMP Gold

There are several RAMP projects which use FPGAs in various ways for simulation, but the most recent and relevant is the RAMP Gold [48] project. This uses an FPGA to perform cycle accurate simulation of multicore and manycore CMP architectures, and is able to simulate up to 64 cores on a relatively cheap Xilinx Virtex-5 FPGA. The two main in-FPGA components are a functional model of the cores, which time multiplexes the multiple instances of the simulated core, sharing memory and cache resources between the models, and a separate timing model. The timing model performs the micro-architectural simulation for each of the simulated cores, and the timing model of the memory hierarchy (i.e. tags only, no data). The timing model drives the functional model's scheduler so that threads are scheduled in the correct order, but because data is only actually cached in a single cache shared by all models, RAMP Gold cannot simulate memory constancy models which violate sequential consistency.

In functional-only mode, RAMP Gold achieves a throughput of up to 100 MIPS when the number of target cores can cover the functional pipeline depth. Like RAMP Gold when functional simulation is compared to the peak performance of software-only simulation the performance of the FPGA architecture simulation is disappointing. However the close to 50 MIPS [48] performance for a 64-core, cache coherent, cycle accurate simulation is currently beyond the limits of software only simulation.

8.5.3 FAST

Another FPGA based simulation system, the FAST project [2,9,10], is one of the fastest cycle accurate simulators that can really claim to be cycle accurate. It does this by performing functional simulation of each core in a high speed functional only simulator (QEMU [40]), and feeding the resulting instruction trace into a high speed timing model, implemented in a FPGA. The FPGA model simulates the pipeline for each core, along with the memory hierarchy, and contains the true memory state of the system, or Oracle Memory as the authors refer to it, from which the software models use cached regions to simulate ahead.

To enable full decoupling of the functional simulations and the timing model, checkpointing snapshots are used, with speculative run-ahead and roll-back, to ensure timing-accurate functional correctness from the functional simulation.

Unfortunately, like the other FPGA based simulators, FAST has not been demonstrated above 64 cores, and total system simulation size is limited by the size of the available FPGA.

8.6 GPGPU Simulation

One of the most interesting parallel simulators, is the CUDA based GPGPU simulation of ARM multicore processors by Pinto et al. [49]. This simulator uses a Nvidia GPU, running a Cuda kernel which implements the instruction set simulator, cache model, and simple interconnect model. Despite executing the simulation in lockstep, it provides simulation rates up to 1800 MIPS for a 8192-core simulation. When adding a

simple switch arbitration model the simulation rate drops to approximately 1 MIPS for 32-core simulation up to 50 MIPS for a 4096-core simulation, with single instruction synchronisation.

Although it currently only supports a subset of the ARM ISA and functional simulation, a large NoC model is highly amenable to GPGPU acceleration. Communication bottlenecks are too high to currently run the core simulation on the CPU and interconnect on the GPU, but moving the whole simulation onto GPU architectures could be a promising way forward for cycle accurate manycore simulators in the future.

9 Conclusion

Having demonstrated simulation speeds of up to 377 MIPS, with an average execution time error of only 2.1% relative to hardware reference implementations, the presented simulator clearly provides a flexible, powerful tool for embedded systems development. With unrivalled performance for software based, multicore, full system, cycle accurate simulation the simulator is not only useful for application development (for timing accurate functional behaviour, debugging, and performance evaluation) but also large scale design space exploration. This was leveraged to perform over 20,000 simulations on a shared cluster service in under three weeks.

This was achieved through novel exploitation of the cache-incoherent nature of embedded systems to decouple simulation components with significant slack, allowing for efficient parallelism. The performance potential was only fully realisable through the presented packet tracking and counting optimisation techniques, which, along with the cache efficient interconnect model and cycle skipping techniques, allows the NoC based interconnect simulation to keep up with the high speed parallel core simulations.

We have also shown that by adding efficient user-space multi-threading these same interconnect modelling optimisations can enable efficient high-speed simulation, including detailed traffic and energy statistics collection, of very large scale coherent systems with embedded processor cores. Examples might be found on a high-compute-density parallel-accelerator or future data-centre processor, and such a simulator would be invaluable in their design.

Our approach is generally applicable to other architectures, although extreme care must be taken with modelling of speculative, superscalar, and simultaneous multi-threading processors, which can continue execution while multiple memory requests are pending. These will usually have to synchronize more frequently with the interconnect model, and as such would benefit from the userspace-context switching introduced in the coherent simulation section.

There has been futher work with both the cache-incoherent and coherent simulators, including techniques for optimising the simulation of synchronising workloads on multi and manycore systems. Along with architectural and microarchitectural innovations explored using the simulation infrastructure, there is also demonstration of the sort of machine-learning prediction that can be enabled with the large datasets that can be rapidly generated by such fast simulators. These can be found in the PhD thesis which developed from this work [16].

References

1. Texas Instruments: DRA72x Infotainment Applications Processor Silicon Revision 2.0 Datasheet. http://www.ti.com/lit/ds/symlink/dra726.pdf (2017)
2. Chiou, D., Angepat, H., Patil, N., Sunwoo, D.: Accurate functional-first multicore simulators. IEEE Comput. Archit. Lett. **8**, 64–67 (2009)
3. ARM: AMBA AXI Protocol Specification, March (2004)
4. The University of Edinburgh, PASTA-2 Receives £1.2m Funding from EPSRC. http://www.icsa.informatics.ed.ac.uk/compilers/news_20104010.html (2010)
5. Chen, J., Annavaram, M., Dubois, M.: SlackSim: a platform for parallel simulations of CMPs on CMPs. SIGARCH Comput. Archit. News **37**, 20–29 (2009)
6. Böhm, I., Franke, B., Topham, N.: Cycle-accurate performance modelling in an ultra-fast just-in-time dynamic binary translation instruction set simulator. In: 2010 International Conference on Embedded Computer Systems (SAMOS), pp. 1–10 (2010)
7. Almer, O., Bohm, I., von Koch, T., Franke, B., Kyle, S., Seeker, V., Thompson, C., Topham, N.: Scalable multi-core simulation using parallel dynamic binary translation. In: 2011 International Conference on Embedded Computer Systems (SAMOS), pp. 190 –199 (2011)
8. Lee, J., Kim, J., Jang, C., Kim, S., Egger, B., Kim, K., Han, S.: Facsim: a fast and cycle-accurate architecture simulator for embedded systems. In: Proceedings of the 2008 ACM SIGPLAN-SIGBED conference on Languages, Compilers, and Tools for Embedded Systems, LCTES '08, pp. 89–100. ACM, New York, NY (2008)
9. Chiou, D., Sunwoo, D., Kim, J., Patil, N.A., Reinhart, W., Johnson, D.E., Keefe, J., Angepat, H.: Fpga-accelerated simulation technologies (fast): fast, full-system, cycle-accurate simulators. In: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 40, pp. 249–261. IEEE Computer Society, Washington, DC (2007)
10. Chiou, D., Sunwoo, D., Angepat, H., Kim, J., Patil, N., Reinhart, W., Johnson, D.: Parallelizing computer system simulators. In: IPDPS 2008. IEEE International Symposium on Parallel and Distributed Processing, pp. 1 –5 (2008)
11. Monchiero, M., Ahn, J.H., Falcón, A., Ortega, D., Faraboschi, P.: How to simulate 1000 cores. SIGARCH Comput. Archit. News **37**, 10–19 (2009)
12. Accellera: TLM-2.0 Reference Manual. http://www.accellera.org/downloads/standards/systemc (2009)
13. T.E.M.B. Consortium: MultiBench 1.0 Multicore Benchmark Software (2010)
14. Mello, A., Maia, I., Greiner, A., Pecheux, F.: Parallel simulation of systemc tlm 2.0 compliant MPsoc on SMP workstations. In: Design, Automation Test in Europe Conference Exhibition (DATE), pp. 606–609 (2010)
15. Butko, A., Garibotti, R., Ost, L., Sassatelli, G.: Accuracy evaluation of GEM5 simulator system. In: 2012 7th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), pp. 1 –7 (2012)
16. Thompson, C.: On the Simulation and Design of Manycore CMPs. PhD thesis, The University of Edinburgh (2015)
17. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 programs: characterization and methodological considerations. In: Proceedings of the 22nd annual Int'l Symposium on Computer Architecture, ISCA '95, pp. 24–36. ACM, New York, NY (1995)
18. Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hållberg, G., Högberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A full system simulation platform. Computer **35**, 50–58 (2002)
19. Lantz, R.: Fast functional simulation with parallel Embra. In: Proceedings of the 4th Annual Workshop on Modeling, Benchmarking and Simulation (2008)

20. Lantz, R.: Parallel SimOS: Scalability and performance for large system simulation. www-cs.stanford.edu (2007)
21. Wang, K., Zhang, Y., Wang, H., Shen, X.: Parallelization of IBM Mambo system simulator in functional modes. ACM SIGOPS Oper. Syst. Rev. **42**(1), 71–76 (2008)
22. Sui, X., Wu, J., Yin, W., Zhou, D., Gong, Z.: MALsim: A functional-level parallel simulation platform for CMPs. In: 2nd International Conference on Computer Engineering and Technology (ICCET), vol. 2, p. V2. IEEE (2010)
23. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D., Wood, D.A.: The gem5 simulator. SIGARCH Comput. Archit. News **39**, 1–7 (2011)
24. Martin, M.M.K., Sorin, D.J., Beckmann, B.M., Marty, M.R., Xu, M., Alameldeen, A.R., Moore, K.E., Hill, M.D., Wood, D.A.: Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. SIGARCH Comput. Archit. News **33**, 92–99 (2005)
25. The edinburgh compute and data facility (ECDF): http://www.ecdf.ed.ac.uk (2015)
26. Zhu, X., Qin, W., Malik, S.: Modeling operation and microarchitecture concurrency for communication architectures with application to retargetable simulation. IEEE Trans. Very Large Scale Integr. VLSI Syst. **14**, 707–716 (2006)
27. Burger, D., Austin, T.M.: The SimpleScalar tool set, version 2.0. SIGARCH Comput. Archit. News **25**, 13–25 (1997)
28. Zhong, R., Zhu, Y., Chen, W., Lin, M., Wong, W.-F.: An inter-core communication enabled multi-core simulator based on simplescalar. In: International Conference on Advanced Information Networking and Applications Workshops, vol. 1, pp. 758–763 (2007)
29. Accellera: SystemC. http://www.accellera.org/downloads/standards/systemc (2015)
30. Hardavellas, N., Somogyi, S., Wenisch, T.F., Wunderlich, R.E., Chen, S., Kim, J., Falsafi, B., Hoe, J.C., Nowatzyk, A.G.: SimFlex: a fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. SIGMETRICS Perform. Eval. Rev. **31**, 31–34 (2004)
31. Zheng, G., Kakulapati, G., Kalé, L.V.: BigSim: A parallel simulator for performance prediction of extremely large parallel machines. In: International Parallel and Distributed Processing Symposium, vol. 1, p. 78 (2004)
32. Kanaujia, S., Papazian, I.E., Chamberlain, J., Baxter, J.: FastMP: a multi-core simulation methodology. In: Proceedings of the Workshop on Modeling, Benchmarking and Simulation (MoBS 2006), Boston, MA (2006)
33. Reinhardt, S.K., Hill, M.D., Larus, J.R., Lebeck, A.R., Lewis, J.C., Wood, D.A.: The wisconsin wind tunnel: virtual prototyping of parallel computers. In: Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '93, pp. 48–60. ACM, New York, NY (1993)
34. Mukherjee, S.S., Reinhardt, S.K., Falsafi, B., Litzkow, M., Hill, M.D., Wood, D.A., Huss-Lederman, S., Larus, J.R.: Wisconsin Wind Tunnel II: a fast, portable parallel architecture simulator. IEEE Concurr **8**, 12–20 (2000)
35. Ren, P., Lis, M., Cho, M.H., Shim, K.S., Fletcher, C., Khan, O., Zheng, N., Devadas, S.: HORNET: a cycle-level multicore simulator. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **31**, 890–903 (2012)
36. Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '05, pp. 190–200. ACM, New York, NY (2005)
37. Kahng, A.B., Li, B., Peh, L.-S., Samadi, K.: ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE '09, pp. 423–428. European Design and Automation Association, Leuven, Belgium (2009)
38. Skadron, K., Stan, M.R., Sankaranarayanan, K., Huang, W., Velusamy, S., Tarjan, D.: Temperature-aware microarchitecture: modeling and implementation. ACM Trans. Archit. Code Optim. **1**, 94–125 (2004)
39. Patel, A., Afram, F., Chen, S., Ghose, K.: MARSS: A full system simulator for multicore x86 CPUs. In: Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE, pp. 1050–1055 (2011)
40. Bellard, F.: QEMU, a fast and portable dynamic translator. In: Proceedings of the 2005 USENIX Annual Technical Conference, ATEC '05, pp. 41–41. USENIX Association, Berkeley, CA (2005)

41. Intel Xeon Phi Product Family: http://intel.com/xeonphi (2015)
42. Miller, J., Kasture, H., Kurian, G., Gruenwald, C., Beckmann, N., Celio, C., Eastep, J., Agarwal, A.: Graphite: a distributed parallel simulator for multicores. In: 2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA), pp. 1 –12 (2010)
43. Carlson, T.E., Heirman, W., Eeckhout, L.: Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulations. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC (2011)
44. Sanchez, D., Kozyrakis, C.: Zsim: fast and accurate microarchitectural simulation of thousand-core systems. In: Proceedings of the 40th Annual International Symposium on Computer Architecture, ISCA '13, pp. 475–486. ACM, New York, NY (2013)
45. Argollo, E., Falcón, A., Faraboschi, P., Monchiero, M., Ortega, D.: COTSon: infrastructure for full system simulation. SIGOPS Oper. Syst. Rev. **43**, 52–61 (2009)
46. Chung, E.S., Papamichael, M.K., Nurvitadhi, E., Hoe, J.C., Mai, K., Falsafi, B.: ProtoFlex: towards scalable, full-system multiprocessor simulations using FPGAs. ACM Trans. Reconfigurable Technol. Syst. **2**, 15:1–15:32 (2009)
47. Spink, T., Wagstaff, H., Franke, B., Topham, N.: Efficient code generation in a region-based dynamic binary translator. In: Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems, LCTES '14, pp. 3–12. ACM, New York, NY (2014)
48. Tan, Z., Waterman, A., Avizienis, R., Lee, Y., Cook, H., Patterson, D., Asanović, K.: RAMP gold: an FPGA-based architecture simulator for multiprocessors. In: Proceedings of the 47th Design Automation Conference, DAC '10, pp. 463–468. ACM, New York, NY (2010)
49. Pinto, C., Raghav, S., Marongiu, A., Ruggiero, M., Atienza, D., Benini, L.: GPGPU-accelerated parallel and fast simulation of thousand-core platforms. In: 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 53–62 (2011)