Scalable, accurate multicore simulation in the 1000-core era

Mieszko Lis* Pengju Ren† Myong Hyon Cho* Keun Sup Shim*
Christopher W. Fletcher* Omer Khan* Srinivas Devadas*
*Massachusetts Institute of Technology, Cambridge, MA, USA †Xi'an Jiaotong University, Xi'an, China

***Abstract*—We present HORNET, a parallel, highly configurable, cycle-level multicore simulator based on an ingress-queued wormhole router NoC architecture. The parallel simulation engine offers cycle-accurate as well as periodic synchronization; while preserving functional accuracy, this permits tradeoffs between perfect timing accuracy and high speed with very good accuracy. When run on 6 separate physical cores on a single die, speedups can exceed a factor of over 5, and when run on a two-die 12-core system with 2-way hyperthreading, speedups exceed 11×.**

Most hardware parameters are configurable, including memory hierarchy, interconnect geometry, bandwidth, crossbar dimensions, and parameters driving power and thermal effects. A highly parametrized table-based NoC design allows a variety of routing and virtual channel allocation algorithms out of the box, ranging from simple DOR routing to complex Valiant, ROMM, or PROM schemes, BSOR, and adaptive routing. HORNET can run in network-only mode using synthetic traffic or traces, directly emulate a MIPS-based multicore, or function as the memory subsystem for native applications executed under the Pin instrumentation tool.

HORNET is freely available under the open-source MIT license at http://csg.csail.mit.edu/hornet/.

I. Introduction

In the recent years, architectures with several distinct CPU cores on a single die have become the standard: general-purpose processors now include as many as eight cores [1] and multicore designs with 64 or more cores are commercially available [2]. Experts predict that by the end of the decade we could have as many as 1000 cores on a single die [3].

For a multicore on this massive scale, connectivity is a major concern. Current interconnects like buses, all-to-all point-to-point connections, and even rings clearly do not scale beyond a few cores. The relatively small scale of existing network-on-chip (NoC) interconnects has allowed plentiful on-chip bandwidth to make up for simple routing [4], but this will not last as scales grow from the 8×8 mesh of a 64-core chip to the 32×32 dimensions of a 1000-core: assuming all-to-all traffic and one flow per source/destination pair, a link in an 8×8 mesh with XY routing carries at most 128 flows, but in a 32×32 mesh, the worst link could be on the critical path of as many as 8,192 flows.

Future multicores will, therefore, require a relatively high-performance network and sophisticated routing. In such complex systems, complex interactions make real-world performance difficult to intuit, and designers have long relied on cycle-level simulations to guide algorithmic and architectural decisions; NoCs are no different. On a multicore scale, however, a cycle-level system simulator has high computation requirements, and taking advantage of the parallel execution capabilities of today's systems is critical.

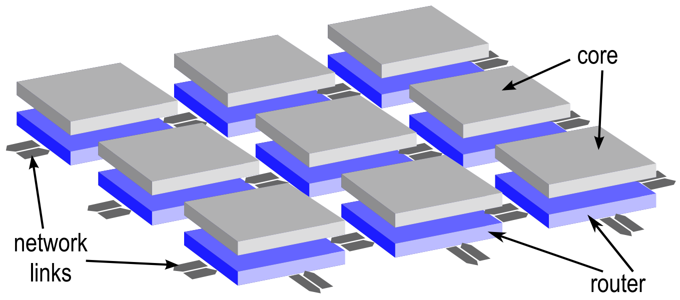


Fig. 1. A multicore system simulated by HORNET. The gray tiles (top) can be trace-driven packet injectors, cycle-level MIPS core models, or threads of an executable run under Pin; the blue tiles (bottom) are cycle-level models of a flit-based virtual-channel wormhole router. While the illustration shows a 2D mesh, HORNET can construct a system with any interconnect geometry.

With this in mind, we present HORNET, a highly configurable, cycle-level multicore simulator with support for a variety of memory hierarchies, interconnect routing and VC allocation algorithms, as well as accurate power and thermal modeling. Its multithreaded simulation engine divides the work equally among available host processor cores, and permits either cycle-accurate precision or increased performance at some accuracy cost via periodic synchronization. HORNET can be driven in network-only mode by synthetic patterns or application traces, in full multicore mode using a built-in MIPS core simulator, or as a multicore memory hierarchy using native applications executed under the Pin instrumentation tool [5].

Specifically, using HORNET, we

- show that results from small-scale NoC simulations cannot be used to guide architectural decisions on a 1000-core scale;
- identify key factors for parallelizing NoC simulators and show how to take advantage of them for linear performance scaling as the number of host cores grows;
- show that without cycle-level simulation, and, in particular, accurate modeling of congestion, various properties of the NoC being simulated (e.g., packet latencies) can suffer significant (e.g., 2×) errors in measurement, and that the detailed information provided by cycle-level simulation can drive architectural decisions;
- demonstrate that end-to-end integration with a processor

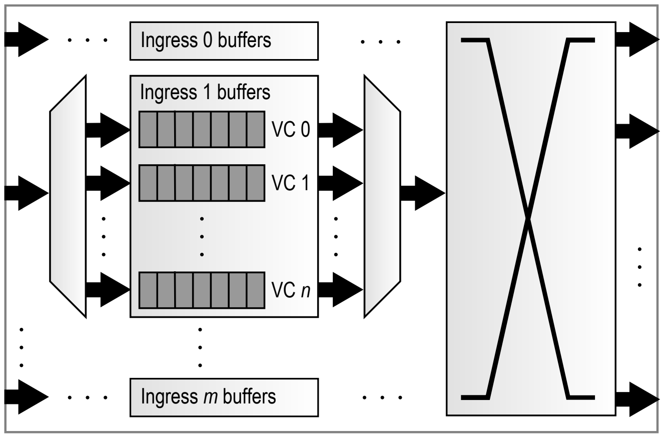


Fig. 2. Basic datapath of an NoC router modeled by HORNET. Packets arrive flit-by-flit on ingress ports and are buffered in ingress virtual channel (VC) buffers until they have been assigned a next-hop node and VC; they then compete for the crossbar and, after crossing, depart from the egress ports.

model is necessary for accurate modeling of application performance;

- describe how simulated power and thermal profiles over the application's runtime and available on a per-tile granularity can drive such decisions as thermal constraint selection and sensor placement, as well as offer opportunities for power-aware routing algorithm design.

In the remainder of the manuscript, we first outline the design and features of HORNET in section II. Next, in section IV, we review the capabilities of HORNET and discuss speed vs. accuracy tradeoffs using complete runs of selected SPLASH-2 applications [6] as well as simulations using synthetic traffic patterns. Finally, we review related research in section V and offer concluding remarks in section VI.

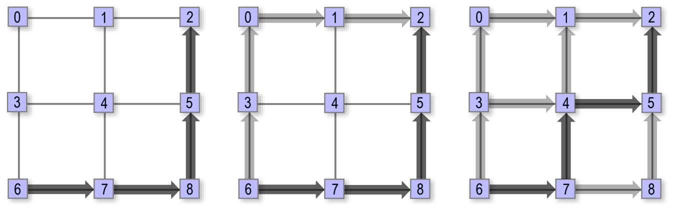
II. Design and features

In this section, we outline the range of systems that can be simulated by HORNET, and discuss the techniques used to parallelize simulations.

A. Network model

Figure 2 illustrates the basic datapath of a NoC router modeled by HORNET. There is one ingress port and one egress port for each neighboring node, as well as for each injector (or CPU core) connected to the switch; each ingress port contains any number of virtual channel buffers (VCs), which buffer flits until they can traverse the crossbar into the next-hop node.

As in any ingress-buffered wormhole router, packets arrive at the ingress ports flit-by-flit, and are stored in the appropriate virtual channel buffers. When the first flit of a packet arrives at the head of a VC buffer, the packet enters the route computation (RC) stage and the next-hop egress port is determined according to the routing algorithm. Next, the packet waits in the VC allocation (VA) stage until granted a next-hop virtual channel according to the chosen VC allocation scheme. Finally, in the switch arbitration (SA) stage, each flit



(a) XY (b) O1TURN (c) two-phase ROMM

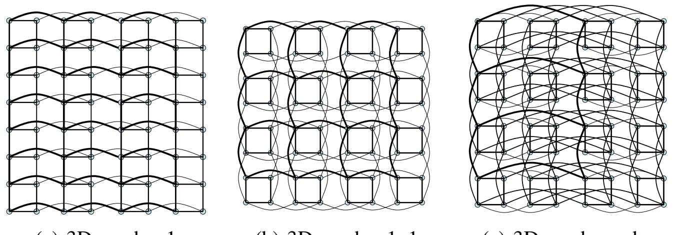
Fig. 3. Example routes for a flow between a source (node 6) and a destination (node 2) for three oblivious routing algorithms. A single path is highlighted in dark gray while other possible paths are shown in light gray.

of the packet competes for access to the crossbar and transits to the next node in the switch traversal (ST) stage. The RC and VA steps are active once per packet (to the head flit), while the SA and ST stages are applied per-flit.

1) Interconnect geometry: The nodes in a system modeled by HORNET can be configured with pairwise connections to form any geometry, including rings, multi-layer meshes (see Figure 4), and tori. Each node may have as many ports as desired: for example, most nodes in the 2D mesh shown in Figure 1 have five ports (four facing the neighboring nodes and one facing the CPU); the number and size of virtual channels can be controlled independently for each port, allowing the CPU$\leftrightarrow$switch ports to have different VC configuration from the switch$\leftrightarrow$switch ports.

2) Routing: HORNET supports oblivious, static, and adaptive routing. A wide range of oblivious and static routing schemes is possible by configuring per-node routing tables. These are addressed by the flow ID and the incoming direction $\langle \mathit{prev\_node\_id}, \mathit{flow\_id} \rangle$, and each entry is a set of weighted next-hop results $\{\langle \mathit{next\_node\_id}, \mathit{next\_flow\_id}, \mathit{weight} \rangle, \cdots\}$. If the set contains more than one next-hop option, one is selected at random with propensity proportionate to the relevant *weight* field, and the packet is forwarded to *next_node_id* with its flow ID renamed to *next_flow_id*.

For example, in the case of simple XY routing, shown in Figure 3a, the routing tables for nodes 2, 5, 6, 7, and 8 would contain one entry for the relevant flow, addressed by the previous node ID (or 6 for the starting node 6) and the flow ID; the lookup result would direct the packet to the next node along the red path (or 2 for the terminal node 2) with the



(a) 3D mesh–x1 (b) 3D mesh–x1y1 (c) 3D mesh–xcube

Fig. 4. Planar view of three example multilayer mesh interconnect geometries which can be directly configured in HORNET.

same flow ID and weight of 1.0. Static routing [7] is handled similarly. For O1TURN routing [8], illustrated in Figure 3b, the table at the start node (6) would contain two next-hop entries (one with next-hop node 3 and the other with next-hop node 7) weighted equally at 0.5, and the destination node (2) would have two entries (one arriving from node 1, and the other from node 5); the remaining tables do not differ from XY.

HORNET's table-driven routing directly supports probabilistic oblivious routing algorithms such as PROM [9]. Probabilistic routing algorithms which first route the packet to a random intermediate node (say via XY routing) and only then to the final destination (e.g., Valiant [10] and its minimum-rectangle variant ROMM [11]). For example, the red path in Figure 3c shows one possible route from node 6 to node 2 in a two-phase ROMM scheme: the packet is first routed to node 4 and then to its final destination. To fill the routing tables, we must solve two problems: (a) remember whether the intermediate hop has been passed, and (b) express several routes with different intermediate destinations but the same next hop as one table entry. The first problem is solved by *changing the flow ID* at the intermediate node, and renaming the flow back to its original ID once at the destination node; the second problem corresponds to sending the flow to one of two possible next-hop nodes weighted by the ratio of possible flows going each way regardless of their intermediate nodes. Consider, as an example, the routing entries at node 4 for a flow from node 6 to node 2. A packet arriving from node 3 must have passed its intermediate hop at node 4 (because otherwise XY routing to the intermediate node would have restricted it to arriving from node 7) and can only continue on to node 5 without renaming the flow. A packet arriving at node 4 from node 7 must not have passed its intermediate node (because otherwise it's out of turns in its second XY phase and it can't get to its destination at node 2); the intermediate node can be either node 1 (with one path) or node 4 itself (also with one path), and so the table entry would direct the packet to node 1 (without flow renaming) or to node 5 (with flow renaming) with equal probability.

3) Virtual channel allocation: Like routing, virtual channel allocation (VCA) is table-driven. The VCA table lookup uses the next-hop node and flow ID computed in the route computation step, and is addressed by the four-tuple $\langle prev\_node\_id, flow\_id, next\_node\_id, next\_flow\_id \rangle$. As with table-driven routing, each lookup may result in a set of possible next-hop VCs $\{\langle next\_vc\_id, weight \rangle, \cdots\}$, and the VCA step randomly selects one VC among the possibilities according to the weights.

This directly supports dynamic VCA (all VCs are listed in the result with equal probabilities) as well as static set VCA [12] (the VC is a function of on the flow ID). Most other VCA schemes used to avoid deadlock, such as that of O1TURN (where the XY and YX subroutes must be on different VCs), Valiant/ROMM (where each phase has a separate VC set), as well as various adaptive VCA schemes like the turn model [13], are easily implemented as a function of the current and next-hop flow IDs.

Finally, HORNET supports VCA schemes where the next-hop VC choice depends on the *contents* of the possible next-hop VCs, such as EDVCA [14] or FAA [15].

4) Bidirectional links: HORNET allows inter-node connections to be bidirectional: links can optionally change direction as often as on every cycle based on local traffic conditions, effectively trading off bandwidth in one direction for bandwidth in the opposite direction [16]. To achieve this, each link is associated with a modeled hardware arbiter which collects information from the two ports facing each other across the link (for example, number of packets ready to traverse the link in each direction and the available destination buffer space) and suitably sets the allowed bandwidth in each direction.

5) Avoiding adversarial traffic patterns: The performance of routing and VC allocation algorithms can be heavily affected by the regular nature of the synthetic traffic patterns often used for evaluation: for example, a simple round-robin VCA scheme can exhibit throughput unfairness and cause otherwise equivalent flows to experience widely different delays if the traffic pattern injects flits in sync with the round-robin period. Worse yet, a similarly biased crossbar arbitration scheme can potentially block traffic arriving from one neighbor by always selecting another ingress port for crossbar traversal.

While relatively sophisticated arbitration algorithms have been developed (e.g., *i*SLIP [17]), the limited area and power in an NoC, together with the requirement for fast line-rate decisions, restricts the complexity of arbitration schemes and, consequently, their robustness to adversarial traffic patterns. Instead of selecting one such algorithm, therefore, HORNET employs randomness to break arbitration ties: for example, the order in which next-in-line packets are considered for VC allocation, and the order in which waiting next-in-line flits are considered for crossbar traversal, are both randomized. While the pseudorandom number generators are by default initialized from an OS randomness source, the random seeds can be set by the user when exact reproducibility is required.

B. Power and thermal modeling

To enable power and thermal analysis, HORNET combines a dynamic power model based on ORION 2.0 [18] with a leakage power model; an accurate thermal model uses HOTSPOT 5.0 [19]. At runtime, various system configuration parameters (buffer sizes, port counts, etc.) and statistics (buffer reads/writes, crossbar transits) are passed to the ORION library for on-the-fly power estimation and to HOTSPOT for thermal modeling: this enables not only the usual average and peak power and thermal analysis for the entire chip but also per-tile and per-time-period reporting.

C. Concurrency, synchronization, and correctness

HORNET takes advantage of modern multicore processors by automatically distributing simulation work among the available cores; as we show in Section IV, this results in significant speedup of the simulations.

The simulated system is divided into tiles comprising a single virtual channel router and any traffic generators connected to it (cf. Figure 1), as well as a private pseudorandom number generator and any data structures required for collecting statistics. One execution thread is spawned for each available processor core (and restricted to run only on that core), and each tile is mapped to a thread; thus, some threads may be responsible for multiple tiles but a tile is never split across threads. Inter-thread communication is thus limited to flits crossing from one node to another, and some fundamentally sequential but rarely used features (such as writing VCD dumps).

Functional correctness requires that inter-tile communication be safe: that is, that all flits in transit across a tile-to-tile link arrive in the order they were sent, and that any metadata kept by the virtual channel buffers (e.g., the number of flits remaining in the packet at the head of the buffer) is correctly updated. In multithreaded simulation mode, HORNET accomplishes this by adding two fine-grained locks in each virtual channel buffer—one lock at the tail (ingress) end of the VC buffer and one lock at the head (egress) end—thus permitting concurrent access to each buffer by the two communicating threads. Because the VC buffer queues are the only point of communication between the two threads, correctly locking the ends when updates are made ensures that no data is lost or reordered.

With functional correctness ensured, we can focus on the correctness of the performance model. One aspect of this is the faithful modeling of the parallelism inherent in synchronous hardware, and applies even for single-threaded simulation; HORNET handles this by having a positive-edge stage (when computations and writes occur, but are not visible when read) and a separate negative-edge stage (when the written data are made visible) for every clock cycle.

Another aspect arises in concurrent simulation: a simulated tile may instantaneously (in one clock cycle) observe a set of changes effected by another tile over several clock cycles. A clock-cycle counter is a simple example of this; other effects may include observing the effects of too many (or too few) flit arrivals and different relative flit arrivals. A significant portion of these effects is addressed by keeping most collected statistics with the flits being transferred and updating them on the fly; for example, a flit's latency is updated incrementally at each node as the flit makes progress through the system, and is therefore immune to variation in the relative clock rates of different tiles. The remaining inaccuracy is controlled by periodically synchronizing all threads on a barrier. 100% accuracy demands that threads be synchronized twice per clock cycle (once on the positive edge and once on the negative edge), and, indeed, simulation results in that mode precisely match those obtained from sequential simulation. Less frequent synchronizations are also possible, and, as discussed in Section IV, result in significant speed benefits at the cost of minor accuracy loss.

Characteristic	Configuration
Topology	32×32 2D mesh, 8×8 2D mesh
Routing	XY, O1TURN, ROMM
VC allocation	dynamic, EDVCA
Link bandwith	1 flit/cycle
VCs per port	4, 8
VC buffer size	4, 8 flits
Avg. packet size	8 flits
Traffic workloads	transpose, bit-complement, shuffle, H.264 decoder profile; SPLASH-2 traces: FFT, RADIX, SWAPTIONS, WATER; natively executed PARSEC applications: BLACKSCHOLES
Warmup cycles	200,000 for synthetic traffic; 0 for applications
Analyzed cycles	2,000,000 for synthetic traffic; full running time for applications
Server CPUs used	2× Intel Xeon X5680 6-core with HT; Intel Core i7 4-core with HT
# HT cores used for simulation	1...24
Sync period	clock-accurate, every 5 cycles

TABLE I
SYSTEM CONFIGURATIONS USED IN SIMULATION

D. Processor core integration

Figure 1 shows the system simulated by HORNET. Each tile contains a flit-based NoC router, connected to other routers via point-to-point links with any desired interconnect geometry, and, optionally, one of several possible traffic generators. These can be either trace-driven injectors, cycle-level MIPS simulators, or instrumented threads of a native application running under Pin. A common bridge abstraction presents a simple packet-based interface to the injectors and cores, hiding the details of DMA transfers and dividing the packets into flits and facilitating the development of new core types.

1) Trace-driven injector: The simple trace-driven injector reads a text-format trace of the injection events: each event contains a timestamp, the flow ID, packet size, and possibly a repeat frequency (for periodic flows). The injector offers packets to the network at the appropriate times, buffering packets in an injector queue if the network cannot accept them and attempting retransmission until the packets are injected. When packets reach their destinations they are immediately discarded.

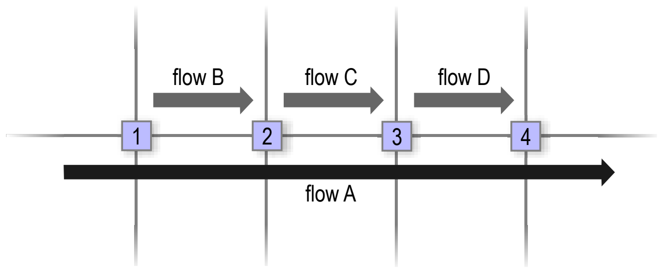


Fig. 5. In a heavily loaded network, traffic on long-path routes can suffer significantly more latency than those on short routes. In this case, flow *A* must compete for the link with a short flow (*B*, *C*, and *D*) at *every step* of its route; assuming locally fair arbitration between any two flows, this effect can result in delays for *A* that are exponential in its path length.

2) MIPS simulator: Each tile can be configured to simulate a built-in single-cycle in-order MIPS core; the core can be loaded with statically linked binaries compiled with a MIPS cross-compiler such as GCC.

The MIPS core is connected to a configurable memory hierarchy which supports an arbitrary number of private or shared cache levels backed by a shared main memory. Memory coherence among the caches is ensured either by an implementation of the MSI cache coherence protocol or via a NUCA-style distributed shared memory with remote-access reads and stores; either option uses the configured on-chip network to communicate with main memories, directories, and other caches.

To directly support MPI-style applications, the network can also be directly exposed to the processor core via a system call interface: the program can send packets on specific flows, poll for packets waiting at the processor ingress, and receive packets from specific queues. The sending and receiving process models a DMA, freeing the processor while the packets are being sent and received.

3) Pin-based native binary instrumentation: HORNET can also be used to instrument native x86 executables using Pin [5]. In this case, the application of interest is run under Pin, and its threads are mapped 1:1 to the simulated tiles as they are spawned. Each instruction executed by the application is intercepted and its memory accesses handled by the memory hierarchy configured in HORNET; timing consists of a table-driven model for the non-memory portion of each instruction plus the memory access latencies reported by HORNET. In this mode, direct application access to the network is not available, and simulation relies on HORNET's coherent memory hierarchy to generate traffic on the network.

III. Methods

To support our claims with concrete examples, we ran HORNET simulations with various system configurations. The salient configuration features used in various combinations in our experiments are listed in Table I.

PARSEC benchmarks were scaled for 1024 cores and run directly on the integrated MIPS model. SPLASH-2 traces were obtained by running the benchmarks [6] in the distributed x86 multicore simulator Graphite [20] with 64 application threads; all network transmissions were logged and the traces were then replayed in HORNET. To obtain significant network congestion, the x86 core was assumed to run on a clock ten times faster than the network. This was necessary because the SPLASH benchmarks were written for a multiprocessor environment where the cost of inter-processor communication was much higher, and thus particular attention was paid to frugal communication; the plentiful bandwidth and relatively short latencies available in NoC-based multicores make this kind of optimization less critical today.

Although each simulation collects a wide variety of statistics, most reports below focus on average in-network latency of delivered traffic—that is, the number of cycles elapsed from the time a flit was injected into a network router ingress port to the time it departed the last network egress port for the destination CPU—as most relevant to current and future cache-coherent shared-memory NoC multicores. For speedups, we measured elapsed wall-clock times with HORNET as the only significant application running on the relevant server. Finally, to quantify the accuracy of the loosely synchronized simulations, we first ran HORNET with full clock-accurate synchronization to obtain a baseline; we then repeated the experiment with different synchronization periods (but the same random number seed etc.) and compared the reported average latencies as an accuracy measurement.

To enable power and thermal analysis, we integrated HORNET with a power model based on ORION 2.0 [18] and a thermal model uses HOTSPOT 5.0 [19].

IV. Discussion

A. Simulation challenges for large-scale multicores

Scaling multicores and their on-chip networks to thousand-core levels presents challenges that do not arise in existing systems with fewer than one hundred cores. On the one hand, there is the simple challenge of significantly more traffic concentrated on few nodes: the off-chip bandwidth grows much more slowly than on-chip transistor counts [21], and the resulting higher core-to-memory ratio will raise traffic centered around the memory controller to unprecedented levels.

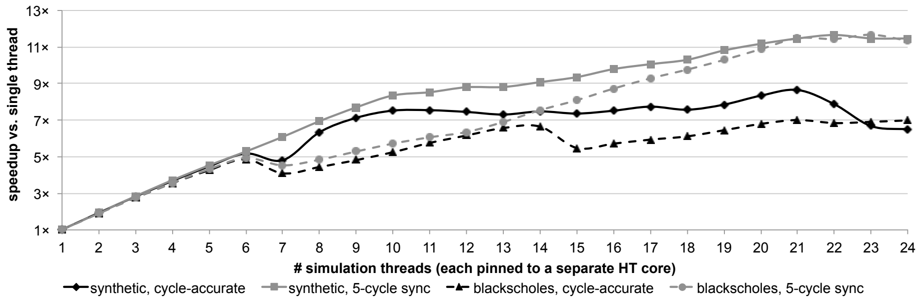
On the other hand, various congestion effects present but not significantly detrimental in smaller networks are radically amplified in on-chip interconnects on a 1000-core scale. For example, while a single one-way link in an 8×8 mesh with domain-ordered routing (DOR) might at worst be the bottleneck for 128 distinct flows (assuming all-to-all traffic and one flow per source/destination node pair), the most encumbered link in a 1024-core 32×32 mesh could be on the critical path of as many as 8,192 flows.[1] Worse yet, local congestion can cause long-distance flows to experience exponentially long latencies (see Figure 5): indeed, in long-running high-traffic simulations of a 1024-core, 32×32 mesh network, we observed that some flows delivered very few or even no packets precisely because of this effect, whereas in a 64-core, 8×8 network this was never a problem.

Clearly, extrapolating architectural decisions for large-scale on-chip networks from small-scale simulations runs severe risks of missing significant performance bottlenecks, and accurate simulation of large networks is a necessary step in the design process.

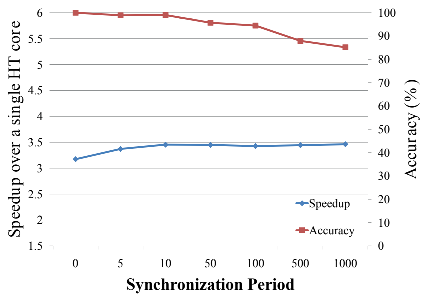
B. Parallelization and performance

Since large-scale designs must be simulated directly, scalability is a key consideration in simulator design. One possibility is to abstract away detail and give up cycle-level simulation, but, as discussed is Section IV-C below, this is undesirable for on-chip network design. Another is to implement the system in FPGA directly or via a time-multiplexing system

[1]the number of flows on the most encumbered link for DOR on an $n \times n$ mesh is $\frac{n^3}{4}$.

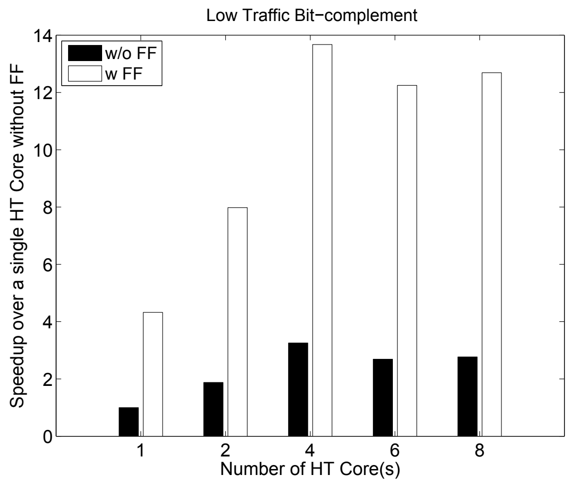


(a) speedup

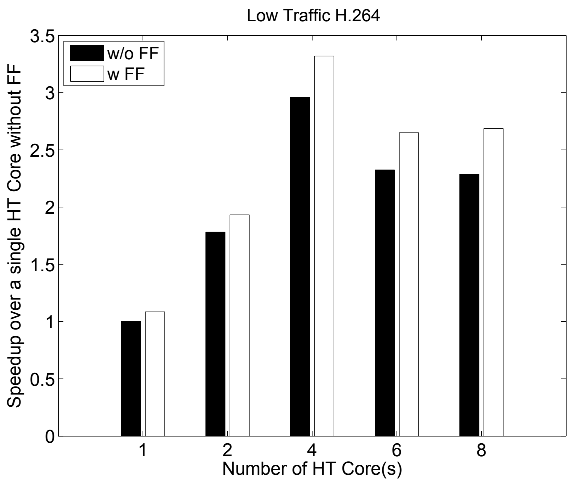


(b) accuracy

Fig. 6. Parallelization speedup for cycle-accurate and loosely-synchronized simulations of 1024-core simulations of synthetic SHUFFLE traffic and the PARSEC BLACKSCHOLES benchmark running in the MIPS frontend on a system with 24 hyperthreaded cores (left) shows that even cycle-accurate simulation scales linearly up to the 6 physical cores on the same die (over 5× speedup). Pairing up threads using HT on the same physical cores offers more speedups for synthetic traffic (which has small cache requirements) than for the MIPS code (simulating which requires more cache); on the other hand, crossing over to the other processor die significantly increases inter-thread synchronization costs, and only becomes advantageous when threads are synchronized loosely. At the same time, simulation fidelity (as measured by average packet latency deviation from cycle-accurate simulation of the synthetic traffic) is near 100%. (On the left, we used a 12× Intel® Xeon® X5680 @ 3.33 GHz, on two 6-core dies, each hyperthreaded 2-way for a total of 24 cores (cores 1–12 on core 0, with 7–12 virtual; cores 13–24 on core 1 with 19–24 virtual); on the right, single-die 4-core Intel® Core™ i7 960 @ 3.2 GHz with 4 threads).



(a) bit complement



(b) H.264 profile

Fig. 7. Performance benefits from fast-forwarding. Unlike low-traffic bit complement, which sends traffic in coordinated bursts and sees significant speedups when the network is idle, the low-traffic H.264 profile gains little because packets are sent relatively constant frequency and the network is rarely fully drained.

like HaSIM [22]; while those approaches offer excellent performance, they require a very low-level, time-consuming design and verification process. In HORNET, we instead take advantage of today's commodity processors featuring several cores on a single die: a single simulation can be split into multiple threads running in parallel on as many cores as are available.

Efficient parallel simulation requires designing the simulator for concurrency from ground up. The key factor limiting performance is inter-thread communication, which can be divided into (a) communication within the simulated network itself, (b) synchronization barriers for clock-accurate results, and (c) any shared data structures; in HORNET, threads share no structures other than queues that carry traffic among the simulated routers, so communication only involves (a) and (b).

Clock synchronization (twice per cycle in cycle-accurate mode) causes the most traffic because all threads must wait on the same barrier. While this is inexpensive and allows linear scaling when all cores are on the same die, barrier communication across separate processor dies becomes time-consuming and limits performance (cf. Figure 6). To allow further speedup, HORNET allows barrier synchronization to be performed instead once every few cycles. From a functional correctness standpoint, this makes no difference, since all traffic will still arrive subject to the original ordering constraints and any deterministic algorithm running on the CPU cores will have the same results. The loose synchronization does imply some loss of fidelity in reported timing, but there HORNET ensures high accuracy by accumulating statistics separately in each thread, carrying measurements within each transmitted

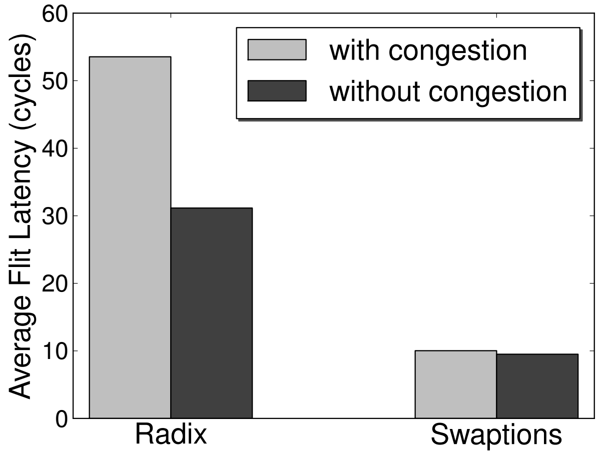


Fig. 8. The effect of congestion on flit latency for two SPLASH applications: for RADIX, which generates a lot of network traffic, not modeling congestion results in a nearly 2× network latency underestimate; for SWAPTIONS, which has much less traffic, the difference, although present, is not significant. The results for the remaining SPLASH applications were similar: the congestion effect for high-traffic applications was similar to RADIX and for low-traffic applications resembled SWAPTIONS. (64-core system with 4 VCs).

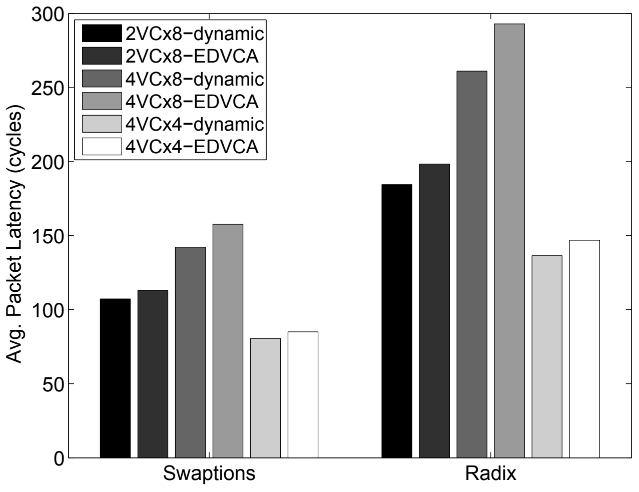


Fig. 9. In-network latency for different VC buffer configurations. Counterintuitively, increasing the number of VCs from 2 to 4 while keeping the VC sizes constant at 8 flits actually *increases* in-network latency because packets can be buffered inside the network. When total VC memory size is held constant, doubling the number of VCs to 4 (and correspondingly halving their capacities to 4 flits) decreases latency as expected. We ran the same experiment on other applications (WATER, and FFT) but the results exhibited the same pattern and so we omit them for brevity.

packet, and never basing measurements on relative values from two cores. As a result, timing measurements retain near 100% fidelity while scaling across separate dies and hyperthreaded cores (Figure 6).

While communication due to simulated network traffic is unavoidable, HORNET employs fine-grained locking to ensure maximum parallelism. The virtual channel buffers—the only communication points between any two tiles—have front and back locks which can be separately held by different threads: this allows HORNET to ensure that results from cycle-accurate parallel simulations are *identical* to those from an equivalent single-thread simulation (given the same randomness seeds), and that intertile communication does not limit performance (cf. Figure 6).

To further improve performance HORNET can fast-forward the clocks in each tile when there are no flits buffered in the network and no flits about to be injected for some period of time. Because in that situation no useful work can possibly result, HORNET advances the clocks to the next injection event and continues cycle-by-cycle simulation from that point without altering simulation results. Clearly, heavy traffic loads will not benefit from fast-forwarding because the network buffers are never drained and HORNET never advances the clock by more than one cycle. Figure 7 shows that the benefit on low-volume traffic depends intimately on the traffic pattern: an application which, like bit complement, has long pauses between traffic bursts, will benefit significantly; an application which spreads the small amount of traffic it generates evenly over time like the H.264 profile will rarely allow the network to fully drain and therefore will benefit little from fast-forwarding.

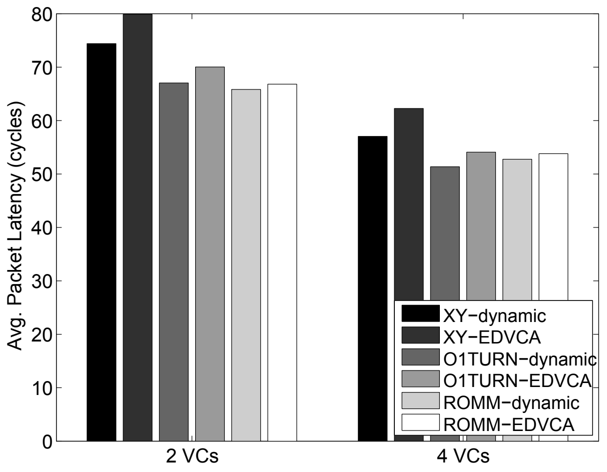


Fig. 10. The effect of routing and VC configuration on network transit latency in a relatively congested network on the WATER benchmark: while O1TURN and ROMM clearly outperform XY, the margin is not particularly impressive. (The remaining application benchmarks are broadly similar except for scaling due to higher or lower network load, and are not shown).

C. Congestion and cycle-level simulation

High-level architectural simulators tend to assume an idealized interconnect network and generally either do not consider congestion or approximate it with an analytical model. For interconnect network design itself, however, congestion effects are of prime importance, as they dictate, for example, what routing algorithms should be employed. To estimate the effect of congestion, we performed simulations of the SPLASH benchmark suite in the congestion-accurate configuration and

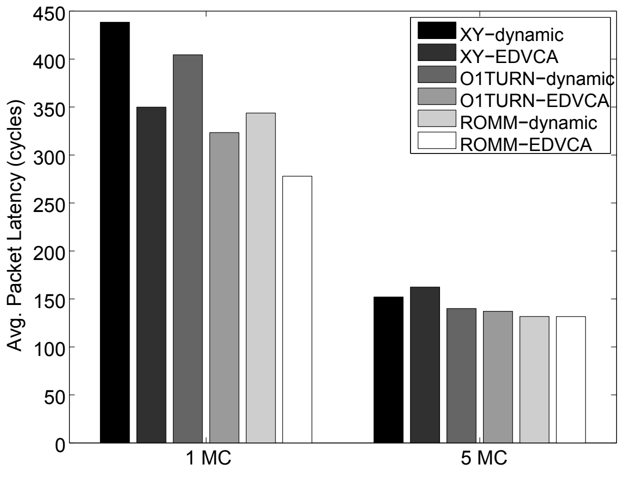


Fig. 11. The effect of varying the number of memory controllers on in-network latency (and therefore memory system performance) running traces from the RADIX benchmark. While multiple memory controllers significantly reduce congestion, replacing one memory controller (1 MC) with five (5 MC) does not increase performance anywhere close to five-fold.

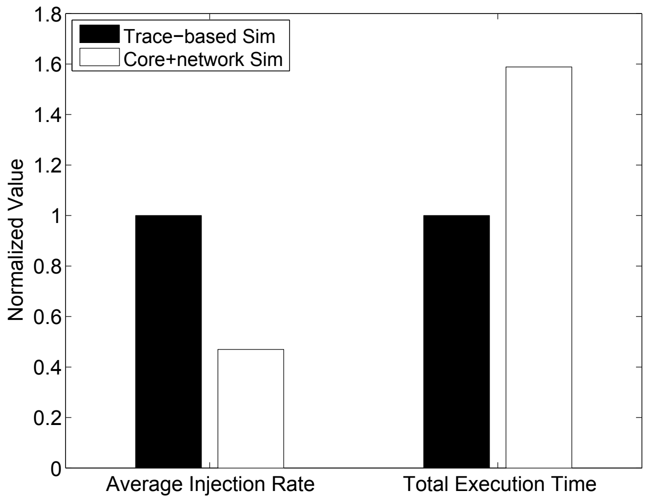


Fig. 12. Trace-based simulation lacks the feedback loop from the network to the sending (or receiving) core; this allows cores to inject packets unrealistically fast and permits the application to finish much earlier than realistically possible.

in a congestion-oblivious configuration where injection bandwidth was limited as in the accurate model but the transit latencies were simple hop-counts. As Figure 8 shows, ignoring congestion effects can cause the simulation to significantly underestimate simulation-time measurements: depending on the amount of network traffic generated by the benchmark, the effect ranged from 2× to negligible.

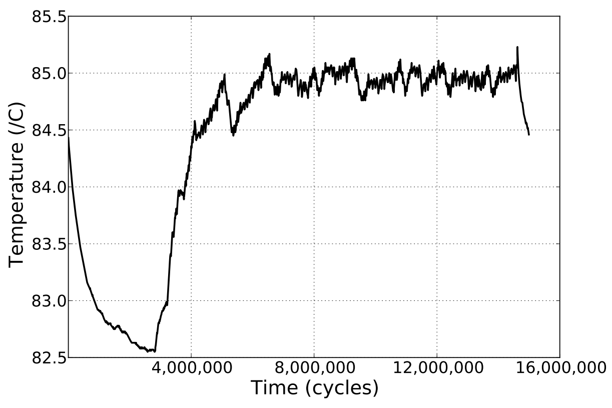
When congestion must be modeled accurately, cycle-accurate simulation is indispensable. For example, network congestion can have significant effects on how the network configuration—say the number and size of the virtual channels—affects in-network latency (i.e., the latency incurred after the relevant flit is seen by the processor as successfully sent). Intuitively, adding more virtual channels should generally allow more packets from different flows to compete for transmission across the crossbar, increase crossbar efficiency, and therefore reduce observed packet latency. While this holds when traffic is light, it may have an opposite effect in a relatively congested network: as Figure 9 illustrates on two SPLASH-2 applications, doubling the number of VCs while holding the size of each VC constant causes the observed in-network latency in a relatively congested network to actually *increase*. This is because the total amount of buffer space in the network also doubled, and, when traffic is heavy and delivery rates are limited by the network bandwidth, the flits at the tails of the VC queues must compete with flits from more VCs and thus experience longer delays. Indeed, when the VC queue sizes are halved to keep the total amount of buffer space the same, the 4-VC setup exhibits shorter latencies than the 2-VC equivalent as originally expected.

While in a lightly loaded network almost any routing and VC allocation algorithm will perform well, heavier loads lead to different congestion under different routing and VC algorithms and performance is significantly affected; again, accurately evaluating such effects calls for a cycle-level simulator and real applications. Figure 10 shows the effect of routing and VC allocation scheme on performance of the SPLASH-2 WATER benchmark in a relatively congested network. While the algorithms with more path diversity (O1TURN and ROMM) do lower observed in-network latency, the performance increase is not as much as might be expected by considering the increased bandwidth available to each flow.

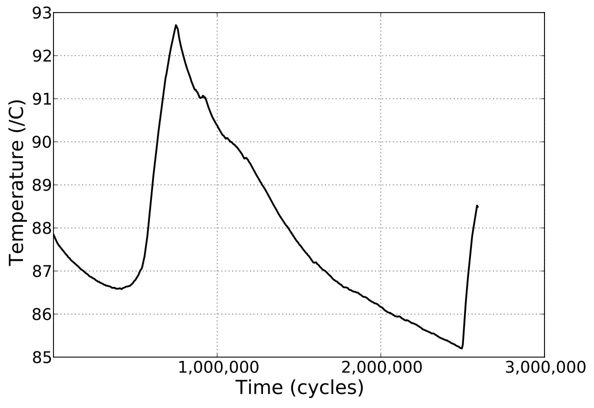
Modern multicore designs can reduce on-chip network congestion by placing several independent memory controllers in different parts of the network. Since in a cache-coherent system a memory controller generally communicates with all processor cores, modeling congestion is critical in evaluating the tradeoff between adding memory controllers and controlling chip area and pin usage. For example, Figure 11 shows in-network latency for two cache-coherent systems: one with one memory controller and the other with five. Although performance clearly improves with five memory controllers, the improvement does not approach five-fold reduction especially for the more congestion-friendly routing and VC allocation schemes. More significantly, the two choices impose different constraints on selecting the routing and VC allocation logic: while the congestion around a central memory controller makes controlling congestion via routing and VC allocation, the average latency in a system with five memory controllers does not vary significantly under different routing algorithms and EDVCA, and the designer might choose to save area and reduce implementation complexity in the network switch.

D. Processor model integration

Much of the research in network-on-chip microarchitecture relies on synthetic traffic patterns or application traces collected under the assumption of an ideal interconnect network.



(a) OCEAN



(b) RADIX

Fig. 13. Temperature traces over the runtime of different SPLASH applications. While for OCEAN a peak (or, indeed, mean) temperature estimate might be used to choose thermal constraints, the activity-dependent temperature variation in radix RADIX means that neither the mean nor the peak provides the best architectural tradeoff.

This approach generally ignores the interdependencies among the various flows and the delays caused by instructions that must wait until network packets are delivered (for example, memory accesses that miss the per-core cache hierarchy). For these reasons, a precise evaluation of the performance of NoC-based multicores in running real applications requires that the CPU core and the network be simulated together.

To quantify the differences between trace-driven and real application traffic, we implemented Cannon's algorithm for matrix multiplication [23] in C using message-passing and targeting the MIPS core simulator that ships with HORNET. We ran the simulation on 64 cores and applied it to a 128×128 matrix; to stress the network, cores were mapped randomly, per-cell data sizes were assumed to be large, and computations were taken to be relatively fast. For the trace version, we assumed an ideal single-cycle network, logged each network transmission event, and later replayed the traces in HORNET; for the combined core+network version, we ran the benchmark with the MIPS cores simulated by HORNET directly interacting with the on-chip network.

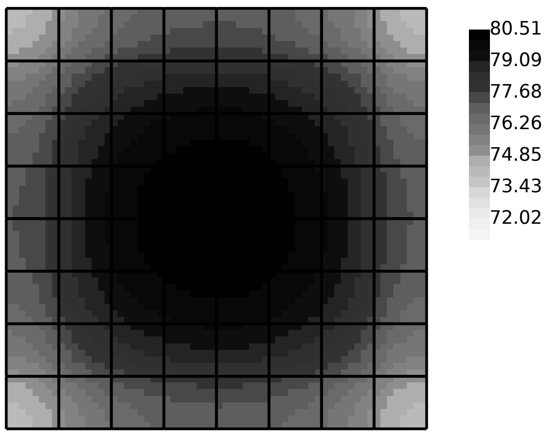
The results, shown in Figure 12, illustrate that the processor cores may have to spend significant amounts of time waiting for the network. On the one hand, a destination node waiting for a packet may block until the packet arrives. On the other hand, the sending node may have to wait for the destination core to make progress: when the destination is nearby (e.g., adjacent), even a relatively short packet can exceed the total buffer space available in the network, and the sending core may have to stall before starting the following packet until the current packet has been at least somewhat processed by the destination core and network buffers have freed up.

E. Thermal effects

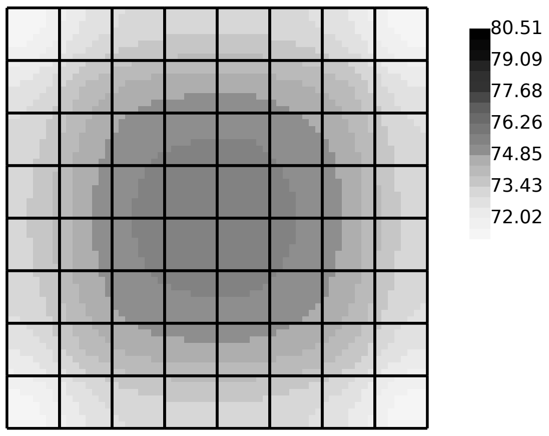
While processor core and cache thermal effects have been extensively studied, available interconnect network models report only steady-state averages for the entire chip. Figure 13 shows that choosing thermal constraints based merely on average or peak temperature data can be misleading: for applications in which network load varies significantly over time, basing interconnect design decisions on the mean values runs the risk of thermal runaways when the application enters a heavy-traffic phase, while using worst-case peak values may result in over-provisioned, expensive thermal packaging. Instead, the designer might choose a design point based on the temperature profiles of the target applications, and ensure that application execution—and hence network traffic—are throttled when temperature rises above some maximum.

Such throttling requires attaching thermal sensors to the die itself; although placing more sensors on the die would provide a more accurate thermal picture, the sensors themselves are relatively expensive and power-hungry, and generally very few are present on a chip. We reasoned that, since our SPLASH runs were done with one memory controller in one corner of the mesh, the switches bordering might become a thermal hotspot and the memory controller would be a good place for a sensor. As illustrated in Figure 14, however, the thermal hotspot in our simulations varied in magnitude but remained in the center of the chip regardless of the benchmark and routing algorithm: this is because the XY routing algorithm we used (and, indeed, nearly all available algorithms) route a greater proportion of the traffic via the central region of the mesh. This result suggests that placing a sensor in the central area of the die should suffice.

The availability of time- and space-resolved thermal measurements within HORNET allows us to investigate routing algorithms which can reroute traffic on possibly longer paths (e.g., near the edges of the mesh) instead of throttling down performance when temperature rises; the development of such power-adaptive routing schemes remains an interesting topic of future research.



(a) RADIX



(b) WATER

Fig. 14. Steady-state temperature distribution over an 8×8 mesh NoC for two SPLASH applications. While the overall magnitude varies significantly (in this case by over 5°C), the overall distribution remains the same: even though the memory controller is located in the lower-left corner, the central nodes suffer the highest temperatures (the remaining SPLASH benchmarks and routing algorithms other than XY show similar temperature profiles).

V. Related work

One NoC simulator that stands out among the many simple, limited-purpose software NoC simulators is Garnet [24]. Like HORNET, Garnet models an NoC interconnection network at the cycle-accurate level: the model allows either a standard ingress-queued virtual channel router with a rigid five-stage pipeline or a flexible egress-queued router. Integration with GEMS provides a full-system simulation framework and a memory model, while integration with ORION [25] provides power estimation. RSIM [26] simulates shared-memory multiprocessors and uniprocessors designed for high instruction-level parallelism; it includes a multiprocessor coherence protocol and interconnect, and models contention at all resources. SICOSYS [27] is a general-purpose interconnection network simulator that captures essential details of low-level simulation, and has been integrated in RSIM. Noxim [28] models a mesh NoC and, like HORNET, allows the user to customize a variety of parameters like network size, VC sizes, packet size distribution, routing scheme, etc.; unlike HORNET, however, it's limited to 2D mesh interconnects and is traffic-pattern-driven rather than integrated with a processor frontend. Booksim [29] allows for more network geometries but is also driven by synthetic traffic patterns. None of these simulators significantly exploit available multicore parallelism.

Highly configurable, parallelized architectural modeling is not a new idea. The Simplescalar toolset [30] can model a variety of processor architectures and memory hierarchies, and enjoys considerable popularity among computer architecture researchers. Graphite [20] is a Pin-based multicore simulator that stands out for its ability to model thousands of cores by dividing the work among not just multiple cores on the same die but multiple networked computers; unlike HORNET's Pin frontend, however, it does not interface with a cycle-level network model and its latency and congestion models are probabilistic. Finally, the growth in complexity and the need for ever-increasing amounts of verification has led to the development of FPGA-based simulators like HaSIM [22] and FPGA-level emulator platforms like RAMP [31], which, though far more difficult to configure, are much faster than software solutions.

VI. Conclusion

We have introduced HORNET, a highly configurable, cycle-accurate network-on-chip simulator that can be driven by network traces, a built-in MIPS simulator, or by native applications instrumented with Pin. HORNET's parallelized simulation engine can scale nearly linearly with the number of physical cores in the processor while preserving cycle-accurate behavior, and allows the user to obtain even more speed via loose synchronization, which preserves correctness but can introduce some inaccuracy in performance measurements.

References

[1] S. Rusu, S. Tam, H. Muljono, D. Ayers, J. Chang, R. Varada, M. Ratta, and S. Vora, "A 45nm 8-core enterprise Xeon® processor," in *Proceedings of the IEEE Asian Solid-State Circuits Conference*, 2009, pp. 9–12.

[2] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64 - processor: A 64-Core SoC with mesh interconnect," in *Proceedings of the IEEE International Solid-State Circuits Conference*, 2008, pp. 88–598.

[3] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the Design Automation Conference*, 2007, pp. 746–749.

[4] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. C. Miao, J. Brown, and A. Agarwal, "On-Chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[5] M. M. Bach, M. Charney, R. Cohn, E. Demikhovsky, T. Devor, K. Hazelwood, A. Jaleel, C. Luk, G. Lyons, H. Patil, and A. Tal, "Analyzing parallel programs with pin," *IEEE Computer*, vol. 43, pp. 34–41, 2010.

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

[7] M. A. Kinsy, M. H. Cho, T. Wen, E. Suh, M. van Dijk, and S. Devadas, "Application-aware deadlock-free oblivious routing," in *Proceedings of the International Symposium on Computer Architecture*. Austin, TX, USA: ACM, 2009, pp. 208–219.

[8] D. Seo, A. Ali, W. Lim, and N. Rafique, "Near-optimal worst-case throughput routing for two-dimensional mesh networks," in *Proceedings of the International Symposium on Computer Architecture*, 2005, pp. 432–443.

[9] M. H. Cho, M. Lis, K. S. Shim, M. A. Kinsy, and S. Devadas, "Path-based, randomized, oblivious, minimal routing," in *International Workshop on Network on Chip Architectures*, 2009, pp. 23–28.

[10] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proceedings of the ACM Symposium on Theory of Computing*. Milwaukee, Wisconsin, United States: ACM, 1981, pp. 263–277.

[11] T. Nesson and S. L. Johnsson, "ROMM routing: A class of efficient minimal routing algorithms," in *Proceedings of the International Workshop on Parallel Computer Routing and Communication*. Springer-Verlag, 1994, pp. 185–199.

[12] K. S. Shim, M. H. Cho, M. A. Kinsy, T. Wen, M. Lis, E. Suh, and S. Devadas, "Static virtual channel allocation in oblivious routing," in *International Symposium on Networks-on-Chip*, 2009, pp. 38–43.

[13] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proceedings of the International Symposium on Computer Architecture*. Queensland, Australia: ACM, 1992, pp. 278–287.

[14] M. Lis, K. S. Shim, M. H. Cho, and S. Devadas, "Guaranteed in-order packet delivery using exclusive dynamic virtual channel allocation," MIT CSAIL, Tech. Rep. MIT-CSAIL-TR-2009-036, 2009.

[15] A. Banerjee and S. Moore, "Flow-aware allocation for on-chip networks," in *Proceedings of the International Symposium on Networks-on-Chip*, 2009, pp. 183–192.

[16] M. H. Cho, M. Lis, K. S. Shim, M. A. Kinsy, T. Wen, and S. Devadas, "Oblivious routing in On-Chip Bandwidth-Adaptive networks," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2009, pp. 181–190.

[17] N. McKeown, "The *i*SLIP scheduling algorithm for input-queued switches," *The IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, 1999.

[18] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2009, pp. 1530–1591.

[19] K. Skandron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *Proceedings of the International Symposium on Computer Architecture*, 2003, pp. 2–13.

[20] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2010, pp. 1–12.

[21] "Assembly and packaging," *International Technology Roadmap for Semiconductors*, 2007.

[22] M. Pellauer, M. Vijayaraghavan, M. Adler, Arvind, and J. Emer, "A-Port Networks: Preserving the Timed Behavior of Synchronous Systems for Modeling on FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, pp. 1–26, 2009.

[23] L. E. Cannon, "A cellular computer to implement the Kalman Filter Algorithm," Ph.D. dissertation, Montana State University, 1969.

[24] N. Agarwal, T. Krishna, L. Peh, and N. Jha, "GARNET: a detailed on-chip network model inside a full-system simulator," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 33–42.

[25] H. Wang, X. Zhu, L. Peh, and S. Malik, "Orion: a power-performance simulator for interconnection networks," in *Proceedings of the International Symposium on Microarchitecture*, 2002, pp. 294–305.

[26] V. S. Pai, P. Ranganathan, and S. V. Adve, "RSIM: Rice Simulator for ILP Multiprocessors," *SIGARCH Comput. Archit. News*, vol. 25, no. 5, p. 1, 1997.

[27] V. Puente, J. Gregorio, and R. Beivide, "Sicosys: An integrated framework for studying interconnection network performance in multiprocessor systems," *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on*, vol. 0, p. 0015, 2002.

[28] (2010) Noxim, the NoC Simulator. [Online]. Available: http://noxim.sourceforge.net/

[29] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally, *Booksim 2.0 User's Guide*, Stanford University, March 2010.

[30] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[31] Arvind, K. Asanovic, D. Chiou, J. C. Hoe, C. Kozyrakis, S.-L. Lu, M. Oskin, D. Patterson, J. Rabaey, and J. Wawrzynek, "RAMP: Research Accelerator for Multiple Processors — a community vision for a shared experimental parallel HW/SW platform," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-05-1412, Sep 2005.