# FIST: A Fast, Lightweight, FPGA-Friendly Packet Latency Estimator for NoC Modeling in Full-System Simulations

Michael K. Papamichael*, James C. Hoe+ and Onur Mutlu+

*Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
+Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA
papamix@cs.cmu.edu, jhoe@ece.cmu.edu, onur@cmu.edu

## ABSTRACT

FIST (Fast Interconnect Simulation Techniques) is a fast and simple packet latency estimator to replace time-consuming detailed Network-on-Chip (NoC) models in full-system performance simulators. FIST combines ideas from analytical network modeling and execution-driven simulation models. The main idea is to abstractly model each router as a load-delay curve and sum load-dependent delay at each visited router to obtain a packet's latency by tracking each router's load at runtime. The resulting latency estimator can accurately capture subtle load-dependent behaviors of a NoC but is much simpler than a full-blown execution-driven model. We study two variations of FIST in the context of a software-based, cycle-level simulation of a tiled chip-multiprocessor (CMP). We evaluate FIST's accuracy and performance relative to the CMP simulator's original execution-driven 2D-mesh NoC model. A static FIST approach (trained offline using uniform random synthetic traffic) achieves less than 6% average error in packet latency and up to 43x average speedup for a 16x16 mesh. A dynamic FIST approach that adds periodic online training reduces the average packet latency error to less than 2% and still maintains an average speedup of up to 18x for a 16x16 mesh. Moreover, an FPGA-based realization of FIST can simulate 2D-mesh networks up to 24x24 nodes, at 3 to 4 orders of magnitude speedup over software-based simulators.

## Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development—*Modeling methodologies*

## General Terms

Design, Performance, Measurement

## Keywords

Modeling, Network-on-Chip, Full-System Simulation, Performance Models, FPGA


*NOCS '11*, May 1-4, 2011, Pittsburgh, PA, USA


## 1. INTRODUCTION

The increasing number of cores in a chip-multiprocessor (CMP) and the demand for full-system modeling has cast simulation speed and complexity as a major obstacle to simulation-based architectural research. As on-chip interconnects become a vital component of future CMPs, there is increasing demand for fast, complexity-effective NoC models.

Typical stand-alone network simulators, such as Booksim [8] or Orion [29] employ high-fidelity, cycle-accurate models to faithfully capture the detailed behavior of the on-chip interconnect — such as the effects of fine-grain packet interactions. While such a high level of fidelity is desirable when studying a new interconnect in isolation, it becomes prohibitively complex and slow — consuming a large fraction of the total simulation time — in the context of full-system simulations. As a result, it is not uncommon for full-system simulators to compromise accuracy for improvements in simulation speed and reduced complexity [4, 15]. In general, within a full-system simulation framework, it is desirable to have fast, low-complexity interconnect models that closely mimic the behavior of their cycle-accurate counterparts. Furthermore, given the recent increased interest in FPGA-accelerated full-system simulation [6, 7], it is also desirable that these interconnect models can be efficiently implemented as part of a broader FPGA-based simulation framework.

In this paper we present **FIST** (**F**ast **I**nterconnect **S**imulation **T**echniques), a set of fast, complexity-effective techniques to accurately estimate NoC packet latencies in the context of full-system CMP simulations. To avoid the cost and complexity of simulating the NoC in detail, FIST borrows an idea from analytical network modeling where each router in the network is abstractly modeled only as a set of load-delay curves. These per-router load-delay curves can be obtained via offline or online training for a given network configuration and traffic pattern. At runtime, instead of a cycle-level detailed simulation of the NoC's operation, FIST only tracks the load experienced by each router based on the observed traffic. In turn, the latency of a packet is estimated by determining which routers are traversed by the packet and then summing the load-dependent latencies at each visited router.

As a proof-of-concept, we present two variants of a simple FIST-based 2D-mesh model and evaluate them against a cycle-accurate interconnect model used within an existing CMP simulator. In addition, to demonstrate the low hardware complexity and scalability of FIST, we include implementation results for an FPGA-based FIST model that

achieves 3 to 4 orders of magnitude speedup over software-based cycle-accurate NoC models.

The rest of this paper is organized as follows. Section 2 provides background on full-system simulation and Section 3 introduces the FIST NoC modeling approach. In Section 4 we describe our evaluation methodology including the FIST model variants we use in our experiments. Section 5 presents accuracy, performance and hardware speedup results. Finally we discuss related work in Section 6 and conclude in Section 7.

## 2. BACKGROUND

Full-system simulation has become an indispensable tool for studying modern multi-core computing systems with sophisticated caching, interconnect, memory, and I/O subsystems, that run rich software stacks and complex commercial workloads. To avoid complexity and detrimental slowdowns, full-system simulators must strike a balance between accuracy and performance. On the one hand, they need to achieve high simulation speeds to be able to simulate modern workloads consisting of tens of billions of instructions. On the other hand, they need to model the system at sufficient fidelity to not unacceptably compromise timing accuracy and skew overall system performance results (e.g., aggregate system instruction throughput).

The required functionality and simulation fidelity of a network model within a full-system simulation framework is typically much simpler compared to a dedicated network simulator. For example, in its simplest form, the network model of a full-system simulator only needs to "tag" packets with their estimated delay. Moreover, full-system simulators can in many cases tolerate some degree of inaccuracy at submodules, as long as the overall performance trends are not significantly affected. This set of goals and requirements gives a different twist to the problem of NoC modeling and forms the guidelines around which FIST was developed.

## 3. THE FIST APPROACH

At an abstract level, any interconnection network, regardless of topology, can be decomposed into a set of routers that are connected by links. At this abstraction level, all buffering and logic within the network is lumped inside the router, which can be treated as a black box device with a set of input and output ports. Our goal in FIST is to leverage this idea and model the network while still retaining this "black box" abstraction, i.e., without having to deal with and model the complex internal structure, logic and state of a router.

To achieve this, FIST borrows an idea from analytical network modeling, where for a given network configuration and traffic pattern each router is represented as a set of load-delay curves. These load-delay curves relate the load at the input ports of a network router to the average latency of a packet going through this router. During runtime, FIST tracks the load at each router and uses these curves for packet latency estimation.

### 3.1 How FIST Works

**Routers as Curves.** As an example, consider a 2D-mesh network that consists of horizontal and vertical links connecting a grid of routers. Figure 1 shows a few possible ways to decompose this mesh network into "black box" components that can be represented by one or more load-delay curves. In the simplest case each router is mapped to a single

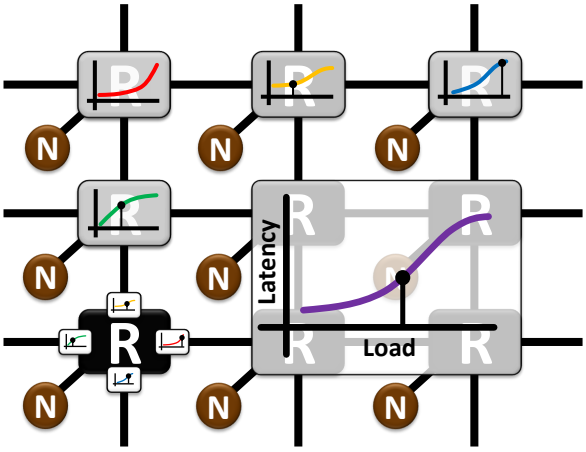



**Figure 1: Representing routers as load-delay curves.**

load-delay curve (as in the top-row routers in the figure), but a more elaborate mapping can be used to adjust the level of abstraction. For example, to raise the level of abstraction, multiple routers can be lumped together and represented by a single curve (as in the four bottom-right routers). Alternatively, if a more detailed representation of the network is desired, then multiple curves can be used for a single router; for instance there can be a separate curve characterizing the traffic for each packet priority class or each output router port (as in the bottom-left router).

To construct these load-delay curves, FIST relies either on an analytical network model or on training that is performed by an existing cycle-accurate network simulator. In the latter case the network simulator can be used for offline or online training. In offline training, the network simulator is used before the actual experiment to pre-generate the curves based on synthetic or actual workload traffic patterns. In online training, the cycle-accurate network simulator occasionally runs on the side and, if needed, can dynamically update the curves while the experiment is running. Online training can still benefit from the existence of a pre-generated set of curves obtained through offline training (these pre-generated curves would be used while the online training warms up).

**FIST in Action.** In the context of full-system simulation, FIST's role is to estimate packet latencies that would be experienced by the same packet in a detailed execution-driven NoC model and deliver packets to their destinations at the appropriate time. To do this, FIST dynamically tracks the load at each router and consults the current set of load-delay curves to estimate packet latencies. For each packet that is injected into the network, FIST follows these four steps:

1. **Routing:** Identify the set of routers that this packet traverses.
2. **Delay query:** Acquire delay estimates at affected routers by indexing their load-delay curves with their current load.
3. **Load update:** Update the load at affected routers to reflect the load increase caused by this packet.
4. **Report total latency:** Sum the partial latency estimates for the affected routers to obtain the total packet latency.

| | NoC Dimensions | | | | | | |
|---|---|---|---|---|---|---|---|
| Datapath | 4x4 | 6x6 | 8x8 | 10x10 | 12x12 | 14x14 | 16x16 |
| 32-bit | 28% | 64% | 114% | 178% | 256% | 348% | 455% |
| 64-bit | 51% | 114% | 202% | 316% | 456% | 620% | 810% |
| 128-bit | 95% | 214% | 380% | 594% | 856% | 1156% | 1521% |
| 256-bit | 184% | 414% | 735% | 1149% | 1655% | 2252% | 2942% |

**Table 1: FPGA resource usage (LUT utilization) for different NoC configurations on a Xilinx Virtex-5 LX330T.**

Contrary to cycle-accurate network simulators, in FIST, injected packets are never actually routed through or stored inside the network model. Instead, they are "instantaneously" processed and tagged with their estimated latency. The only state maintained and updated over time in FIST is the load at each router. By relying on analytical models or high fidelity network models for training, FIST is able to offload a large fraction of the modeling effort and significantly reduce its implementation complexity. The lightweight nature of FIST allows for very efficient software-based or low cost and scalable FPGA-based implementations.

## 3.2 FIST-based Network Models

The "black box" approach of FIST-based network models allows them to be used as building blocks within larger software-based or hardware-based full-system simulation environments. Depending on the implemented routing logic and the selected load-delay curves, FIST can model a variety of network topologies for a range of different network configuration parameters (e.g., number of VCs, buffer sizes, arbitration logic).

**Software-based Implementations.** In the case of software-based full-system simulators, FIST presents a fast and lightweight alternative to existing high-fidelity network models. FIST can either completely replace existing network models or can run on the side in tandem with the existing network models to accelerate less critical regions of a workload. In this latter case, existing network models can interact with FIST to provide feedback on the observed latency estimation error and dynamically update the load-delay curves for online training.

**Hardware-based Implementations.** The simple, lightweight nature of FIST-based network models makes them ideal candidates for hardware implementation on an FPGA. Although FPGAs have traditionally been used as prototyping platforms for networks [26, 25], such approaches are complex and entail high implementation and verification effort, both for the initial design as well as for any subsequent modifications, because they accurately model all components of a router. More importantly, they suffer from limited scalability, as current FPGAs can only emulate small designs due to capacity constraints [28]. To illustrate this point, Table 1 shows the fraction of FPGA area needed for a 2D-mesh NoC that uses a state-of-the-art virtual channel (VC) router [19] for various sizes and datapath widths, obtained using FPGA synthesis. Faded cells represent design points that would not fit on the FPGA. Even on one of the largest Xilinx FPGA parts, the designs that fit on the FPGA are only limited to 4x4 or 6x6 meshes with a narrow datapath. A 16x16 design with a wide datapath would require 29X of the area of such an FPGA.

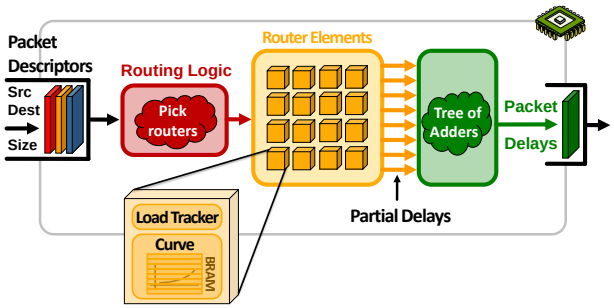



**Figure 2: FIST Hardware Block Diagram.**

**FIST Hardware Architecture.** Figure 2 shows the architectural block diagram for our hardware implementation of FIST targeting FPGAs. Our hardware implementation accepts packet descriptors through a FIFO-based input interface that contain packet information, such as source, destination and size. After being processed, packets are tagged with a latency estimate and are placed in a FIFO-based output interface. The design is pipelined and can process a new packet descriptor every clock cycle.

For each packet, the hardware version of FIST follows the same four processing steps mentioned earlier in Section 2. Once a new packet descriptor is received through the FIFO input, routing logic determines which routers are affected by this packet. The affected routers then all simultaneously perform a parallel lookup in their load-delay curve based on their current load and subsequently update the load to reflect this packet's contribution to additional load. Finally, a tree of adders sums up the partial router latencies to tag the packet with the final latency estimate, at which point it is ready to be placed in the output FIFO interface.

Compared to existing approaches [25, 27, 26] that use FPGAs to implement the target simulated network at cycle-level accuracy, FIST provides an attractive lower fidelity alternative that not only avoids the high implementation complexity, but also vastly increases the scalability of the model, in terms of the number of simulated routers. As shown later in the results section, we were able to fit up to a 24x24 (576-router) 2D-mesh FIST network model on an FPGA that perfectly replicates the software-based FIST models presented in Section 4.

## 3.3 FIST Limitations and Requirements

FIST-based network models are meant to be used within full-system simulation environments that can tolerate or simply accept some degree of simulation inaccuracies. They are not meant to replace dedicated cycle-accurate models built to study networks in detail. In fact, FIST relies on the existence of such detailed network models for the purposes of training. However, even under such conditions, the abstract modeling approach employed by FIST limits its applicability to certain types of networks and specific traffic patterns.

**FIST-friendly Networks.** When used with a static set of load-delay curves obtained through offline training, FIST models can successfully capture the behavior of networks that a) exhibit stable and predictable behavior as loads at individual routers fluctuate and b) are used with traffic patterns that do not significantly deviate from the patterns used during offline training. As a counterexample, networks that react to changes in load or in traffic patterns by drastically altering their routing decisions or arbitration policies are

not good candidates for offline-trained FIST models. For instance, FIST is likely to fail when trying to model an adaptive network that employs GOAL routing [23] or a network where the traffic pattern rapidly oscillates between "uniform random" and "neighbor" [8].

To overcome some of the above limitations, FIST can be used with online training. In this case a cycle-accurate network model is occasionally run in tandem with the FIST model and "re-trains" the load-delay curves of FIST on the fly. Such a hybrid simulation model is able to adapt to transient changes in network behavior and traffic patterns, as long as these changes occur in sufficiently long stable intervals to be captured by the occasional "re-training". Even with online training, however, a network that encounters rapid and vast changes in traffic patterns at a faster rate than the training rate will likely not be faithfully modeled using FIST.

In summary, FIST's limitations stem from its abstract representation of the network using load-delay curves. FIST can be viewed as a predictor that is only as good as the representativeness of its training data. The accuracy of packet latency estimates depends heavily on the fidelity and representativeness of the models used for training. Moreover, since FIST relies on training data from an existing analytical or simulation model, it is obviously not useful for performing exploratory studies that study new types of networks.

**Accuracy Sensitivity to Load and Buffering.** Regardless of the type of training, the accuracy of FIST is also affected by the amount of buffering and the average load in the network. Intuitively, the amount of buffering determines the range of possible packet latencies and the load determines the amount of variance in observed packet latencies. As network load increases, variance is amplified, because of more fine-grain packet interactions (e.g., output contention) and because of congestion that builds up at some routers and eventually starts affecting other neighboring routers.

**FIST for NoC Modeling.** In a modern CMP, NoCs have to share chip resources with other components such as cores, caches and memory controllers. Given the above limitations and the scarcity of on-chip resources, such as area, power and on-chip memory, FIST would be a good candidate for modeling NoCs that are part of a broader CMP system for the following reasons:

- Due to resource contention among different components on a CMP, NoCs are usually relatively simple designs. As an example, two recent industry-developed NoCs [10, 21] employ dimension-ordered routing, one of the simplest deterministic routing algorithms. Such algorithms are easy to model using FIST.

- To compensate for their simple design and provide worst-case guarantees, NoCs are often over-provisioned and, as a result, many on-chip interconnection networks are observed to be operating at low packet injection rates relative to peak. This in turn results in low router loads [11, 13, 17], which reduce packet latency variance and create good conditions for FIST-based latency predictions.

- Given the abundance of wires in on-chip environments which allows for very wide datapaths, buffering in NoCs is usually limited or in some cases completely eliminated [17]. As was the case with low loads, limited buffering also helps FIST in accurate latency estimation, because it limits the range and variance of latency values, especially when combined with low loads.

# 4. EVALUATION

We examine two variants of a simple FIST model for 4x4, 8x8 and 16x16 2D-mesh NoCs that employ DO (Dimension Ordered) wormhole routing. We use 4VCs/channel and use 8, 16 and 32-flit VC buffers. To measure the effectiveness of FIST in a broader simulation environment, we use our FIST models to replace an existing cycle-accurate NoC simulator used within an in-house tiled CMP simulator the front-end of which is based on Pin [5]. Table 2 shows the major system and network parameters.

We run multiprogrammed and multithreaded workloads and observe how FIST affects salient simulation results, such as average packet latency and average aggregate instruction throughput. Traffic is generated due to cache misses and consists of single-flit control packets for data requests or coherence control messages and 8-flit data packets for cache block transfers.

**Workloads.** We use 26 SPEC CPU2006 [24] benchmarks for multiprogrammed workloads, as well as 8 SPLASH-2 [22] and 2 PARSEC [1] multithreaded workloads, which are listed in Table 3. To form multiprogrammed workloads, we classified the SPEC CPU2006 workloads in terms of network intensity and formed multiprogrammed workloads of varying intensity. Each benchmark was compiled using gcc 4.1.2 with -O3 optimizations and a representative simulation phase was chosen using PinPoints [20]. For the 4x4, 8x8 and 16x16 mesh configurations we run workloads for 50, 20 and 5 million cycles respectively, or to completion if they finish earlier than their allotted cycles.

For multiprogrammed workloads, we classify benchmarks based on the amount of L1 MPKI (misses per kilo instructions), which is tightly coupled to their network intensity; benchmarks with an average MPKI greater than ten are labeled as network-intensive, while all other benchmarks are labeled as network-non-intensive. Depending on the fraction of network-intensive workloads, multiprogrammed workloads (MP) are classified as low (0%), medium (50%) or high (100%) intensity. For each memory intensity category (0%, 50% and 100%), we simulate 16 multiprogrammed workloads, for a total of 48 workloads in the case of 4x4 or 8x8 meshes. For 16x16 mesh configurations we only simulate 24 multiprogrammed workloads, 8 from each network intensity class.

**Model Parameters.** In our FIST models we represent each router as a set of two load-delay curves. The first curve relates the load at a router to the average network latency of a packet traversing this router on its way to the destination. This network latency is defined to be the time interval from the moment a packet's tail flit enters a router's VC buffers until the moment it reaches the next router on its path. The second curve relates the load at a router against the injection latency that a packet observes when it is injected into the network from the node attached to that router.

Load, which is used to index the load-delay curves, is defined as the number of flits observed on the five input ports of a router in the last HL (HistoryLength) cycles. We set the HL parameter according to the size of the VC buffer size of the NoC. For a VC buffer size of 8, 16 and 32 flits we set the HL parameter to 64, 128 and 256 cycles respectively.

| | |
|---|---|
| System topologies | 4x4, 8x8, 16x16 mesh (core + coherent L1 + slice of the distributed L2 at each node) |
| CPU model | Out-of-order x86, 128-entry instruction window, 16 MSHRs |
| Private L1 cache | 64 KB, 2-way associative, 64-byte block size |
| Shared L2 cache | Perfect (always hits), distributed (S-NUCA [12]) with randomized static mapping of cache blocks to L2 slices |
| Coherence protocol | Simple directory-based, based on SGI Origin [14], perfect directory |
| Interconnect links | 1-cycle latency, 64-bit flit width (8 flits per cache block) |
| Router configurations | 4-cycle latency, 4VCs/channel, 8 and16 flits/VC for 4x4 and 8x8, 16 flits/VC for 16x16 |

**Table 2: System parameters used in our evaluation.**

| | |
|---|---|
| SPEC2006 Benchmarks for multiprogrammed workloads | **Low network intensity:** gromacs, tonto, calculix, namd, dealII, h264, gobmk, sjeng, hmmer, perlbench **High network intensity:** xalancbmk, gcc, libquantum, sphinx3, bzip2, povray, mcf, lbm, soplex, leslie3d, astar, omnetpp, gems, wrf, milc, cactus |
| Multithreaded workloads | **SPLASH-2:** cholesky, lu_n, lu_c, ocean_n, barnes, fmm, water_n, water_s **PARSEC:** streamcluster, fluidanimate |

**Table 3: Workload Information.**

**Offline and Online FIST.** We examine two variants of the presented NoC model. The first (*FIST_offline)* relies on offline training with synthetic traffic using the original cycle-accurate NoC model. Given the static random mapping of cache blocks to L2 slices, we train using a uniform random synthetic traffic pattern, which we expect to resemble actual workload traffic patterns.

The second variant (*FIST_online*) relies on online training that is occasionally performed by the original cycle-accurate NoC model to obtain and dynamically update the FIST load-delay curves. In this scheme, routers are initially loaded with the same load-delay curves used in the $FIST\_offline$ model. Once simulation starts, the cycle-accurate NoC model is enabled every $Quantum_{cycles}$ cycles and is then used to observe the packet latency error and train FIST for $Train_{cycles}$ cycles. In the beginning of each training period, the cycle-accurate NoC simulator is run for $Warmup_{cycles}$ to warm up its structures. Training continues until the average packet latency drops below $Error_{threshold}$. Unless stated otherwise we set $Error_{threshold}$ to 5%, which means that training will continue until the average packet latency error during the last $Train_{cycles}$ cycles falls below 5%. We empirically set the default $Quantum_{cycles}$, $Train_{cycles}$ and $Warmup_{cycles}$ to 100000, 10000 and 1000 cycles respectively. In the results section we also experiment with more aggressive values that reduce the amount of training to observe the effects on error and performance.

To avoid abrupt changes in predicted latencies during training, updated latency values affect the existing latency values in a decaying fashion, where each new latency sample only contributes by a factor of $Decay_{factor}$ to the existing average latency for a specific load. Given a new latency sample $Lat_{sample}$ observed at some load, the updated latency $Lat_{updated}$ is calculated based on the existing latency $Lat_{prev}$ as follows:

$$Lat_{updated} = \frac{(Decay_{factor} - 1) * Lat_{prev} + Lat_{sample}}{Decay_{factor}}$$

We empirically set the $Decay_{factor}$ to 100, a value that appeared to be high enough to tolerate noise in the latency updates, but also low enough to quickly adapt to changes in workload and traffic behavior.

**Accuracy**. To assess the simulation infidelity introduced by FIST, we present the observed error in average packet latency and average aggregate instruction throughput. Average packet latency is obtained by summing the latencies of all packets and dividing this sum by the number of packets. Average aggregate instruction throughput ($IPC_{\Sigma}$) is defined as the sum of the average IPC (Instructions Per Cycle) for each core in the simulated system. Finally for any metric M we define the error $E$ between the actual value $M_{actual}$ and the estimated value $M_{estimate}$ as follows:

$$E = \frac{M_{estimate} - M_{actual}}{M_{actual}}$$

To gain deeper insight than what average values can provide, we also present a case study where we continuously run both the actual cycle-accurate and the FIST NoC models side-by-side and present a timeline of the actual and the estimated latencies.

**Hardware Evaluation.** To evaluate the cost and performance of an FPGA-based FIST implementation, we developed an RTL model written in Bluespec System Verilog [2] that realizes the architecture discussed in Section 3 and shown in Figure 2. Our implementation precisely replicates the presented software-based model. We show results for FPGA resource usage and post-synthesis clock frequency estimates for a moderately sized (LX155T) and a large (LX760) Xilinx FPGA.

**Performance.** To evaluate performance for the software-based models, we present the approximate speedups observed for the various FIST variants. For the software-based models, we report the network simulation speedup, which was measured based on average wall clock time of hundreds of experiments run on a uniform 256-core cluster based on 2.5GHz Intel Xeon processors. For the FIST models that rely on online training we also report FIST utilization, which is the fraction of cycles that the FIST model was running alone, without the aid of the cycle-accurate model. For the FPGA-based FIST models we report performance based on post-synthesis clock frequency results.

# 5. RESULTS

We first present accuracy results for our software-based FIST models for 4x4, 8x8 and 16x16 mesh configurations. We extend our accuracy results with a case study for a sample multithreaded workload and highlight specific regions of interest. We then present FPGA implementation results and finally report speedup for both the software and hardware FIST models.

## 5.1 Accuracy and Performance Results

Figure 3 plots the error in average packet latency and aggregate IPC across all workloads using the *FIST_offline*

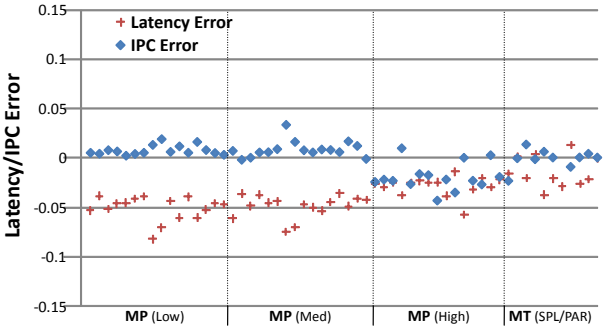



**Figure 3: Average Latency and Aggregate IPC Error for 8x8 mesh using *FIST_offline* model.**

model for an 8x8 network. The error for average latency is in all cases below 10%, whereas $IPC_{\Sigma}$ error is significantly lower, always below 5%. There is a clear bias towards underestimation of latency (i.e., negative error), which can be attributed to using uniform random traffic for training; in our experiments we observed that even though traffic patterns generally resemble uniform random over long periods of execution, workloads tend to go through phases of varying intensity, that create transient hot-spots. In such cases, under the same load, the affected routers will observe higher latencies compared to uniform random traffic.

The same error results, but this time using the *FIST_online* model that leverages dynamic training information, are shown in Figure 4. The benefits of online training are clear; error for both latency and $IPC_{\Sigma}$ are greatly reduced and lie in all cases well within the 5% range. In addition, the underestimation bias is now eliminated; transient workload behavior changes are now captured by the cycle-accurate simulator that occasionally runs on the side and the FIST model is able to successfully adapt.

Table 4 presents Latency and $IPC_{\Sigma}$ error for all experimental configurations, as well as observed network simulation speedup and the fraction of time spent in network simulation when using the cycle accurate NoC model. For experiments that use the *FIST_online* model we also report FIST utilization. Finally, we also report results for a more aggressive variant of *FIST_online* (denoted by $Onl_{aggr}$ in the table) where we set the $Quantum_{cycles}$, $Train_{cycles}$ and $Warmup_{cycles}$ parameters to 500000, 5000 and 1000 cycles respectively. Such a configuration introduces slightly higher error but achieves much better performance, since it is only forced to train 1% of the time, instead of 10%, which is the case for our baseline results with *FIST_online*.

**Effect of Latency Error on IPC Error.** It is interesting

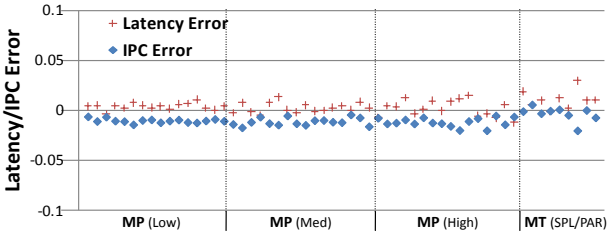



**Figure 4: Average Latency and Aggregate IPC Error for 8x8 mesh using *FIST_online* model.**

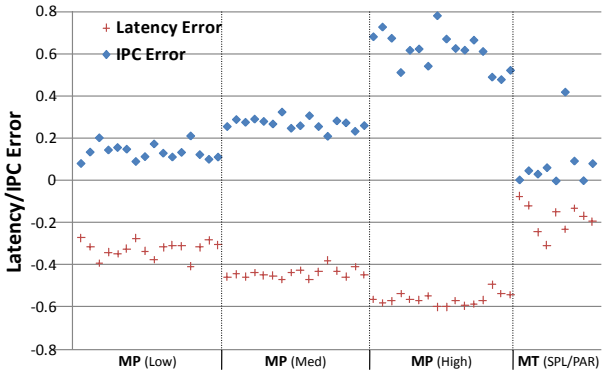



**Figure 5: Average Latency and Aggregate IPC Error for 8x8 mesh using *hop-based* model.**

to note that for both FIST models the aggregate instruction throughput ($IPC_{\Sigma}$) error is significantly lower than the latency error. This indicates that IPC, one of the most commonly reported results in full-system simulations, is stable under small network modeling errors. However, to show that this does not hold when larger network errors are introduced, we also ran experiments with a very simplistic *hop-based* model that estimates packet latencies based on the number of hops between two nodes and does not consider network load at all.

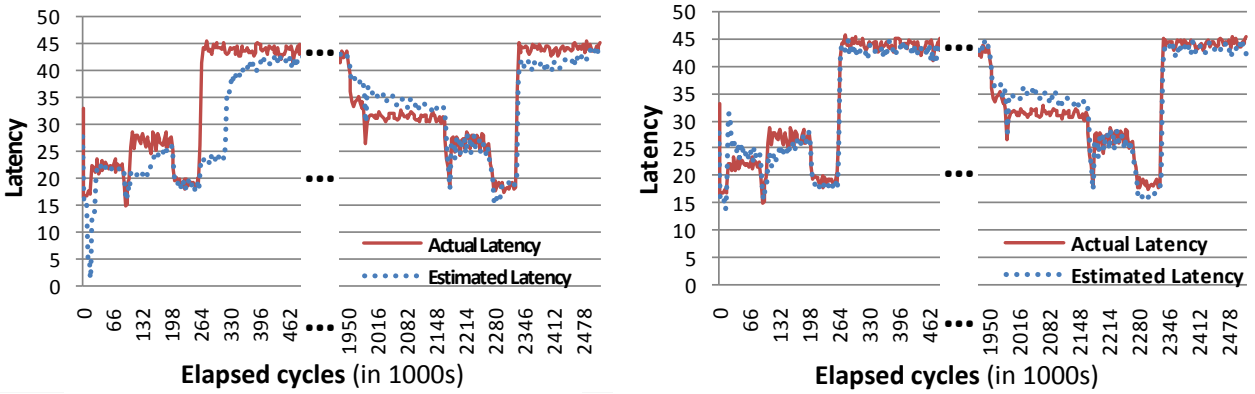
The results for the simple hop-based model are shown in Figure 5. As expected, this model introduces much higher latency error, especially for the more network-intensive workloads, where it reaches up to 60%. Contrary to what was shown in the two previous figures, the severe error in latency does indeed translate into IPC error, which in some cases approaches 80%. These results indicate that full-system simulation might be able to tolerate some amount of infidelity in the network model, but beyond some error threshold, this infidelity can start introducing significant errors in higher-level simulation results.

**Case Study.** To get a glimpse of how FIST works at a fine-grain level and to demonstrate the effects of online training, Figure 6 (a) shows a timeline of average actual and estimated latency sampled every 3000 cycles for a total of 3 million cycles. Due to space limitations, we condense a 1.5M stable phase that would otherwise span the middle of the timeline. To produce the timeline, we used the "LU" SPLASH-2 workload, which exhibits very distinct phases due to barrier synchronizations. For this experiment, we use the *FIST_online* model, but to better highlight the effects of training, we intentionally load it with initial curves that report a latency of zero for all loads. The solid line corresponds to the actual latency reported by the cycle-accurate simulator and the dashed line corresponds to latencies estimated by FIST.

Having no previous training data, FIST initially is unable to immediately capture the sudden latency changes in this workload. Eventually, though, with the help of online training, it approaches the actual latencies reported by the cycle-accurate model. Notice for instance that FIST is initially unable to properly estimate the latency spike around 300K cycles, but can do so the second time it encounters a similar spike around 2300K cycles. The reaction delay exists because the cycle-accurate model is invoked only every $Quantum_{cycles}$. The presence of an initial set of curves used by the *FIST_online* model can alleviate such problems as is

| Mesh size | 4x4 | | | | | | | 8x8 | | | | | 16x16 (for 24 MP workloads) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Flits/VC Buf | 8 | | 16 | | | 32 | | 8 | | 16 | | | 16 | | |
| % in Net. Sim. | 47.4% | | 46.1% | | | 46.4% | | 53.2% | | 51.7% | | | 58.2% | | |
| FIST model | Offl | Onl | Offl | Onl | $Onl_{aggr}$ | Offl | Onl | Offl | Onl | Offl | Onl | $Onl_{aggr}$ | Offl | Onl | $Onl_{aggr}$ |
| Lat. Error % | 3.7 | 1.39 | 5.08 | 1.35 | 1.46 | 5.81 | 1.37 | 1.54 | 0.92 | 3.88 | 0.6 | 0.82 | 4.77 | 1.34 | 1.63 |
| $IPC_{\Sigma}$ Error % | 1.65 | 0.70 | 1.44 | 0.58 | 0.73 | 1.89 | 0.53 | 3.39 | 1.19 | 1.66 | 1.07 | 1.25 | 4.64 | 0.84 | 1.18 |
| Speedup | 13.36 | 3.88 | 14.31 | 4.30 | 7.31 | 12.93 | 4.41 | 22.22 | 5.79 | 26.00 | 6.68 | 16.10 | 43.13 | 7.85 | 18.42 |
| FIST Util. % | - | 81.79 | - | 83.30 | 97.13 | - | 84.91 | - | 84.95 | - | 87.23 | 98.13 | - | 85.66 | 96.50 |

**Table 4: Accuracy, performance and other results for all experiments across 4x4, 8x8 and 16x16 mesh configurations.**




(a) Without initial training data

(b) With initial training data

**Figure 6: Actual and Estimated Latency for "LU" SPLASH-2 benchmark over a window of 3 million cycles.**

shown in Figure 6(b).

## 5.2 Hardware Implementation Results

Our hardware implementation of FIST precisely replicates the presented software-based model. On a relatively small Xilinx FPGA (Virtex-5 LX155T) we were able to fit models for up to 20x20 mesh networks running at 200MHz, which corresponds to a packet processing rate of 200M packets/sec. On a larger, more recent Xilinx FPGA (Virtex-6 LX760) we were able to successfully synthesize designs up to 24x24 running at estimated frequencies beyond 300MHz, which corresponds to 300M packets/sec.

To put these numbers in perspective, for a small 4x4 network, the cycle-accurate software model used in this paper can process packets at an approximate rate of 30K packets/sec, which makes our FIST hardware implementation three to four orders of magnitude faster. Detailed resource usage and post-synthesis clock frequency results are shown in Table 5. The design uses only a small amount of LUTs. As a result, its scalability is only limited by the number of BRAMs required to store the load-delay curves.

| | Virtex-5 LX155T | | | Virtex-6 LX760 | | |
|---|---|---|---|---|---|---|
| Size | BRAMs | LUT % | Freq. | BRAMs | LUT % | Freq. |
| 4x4 | 8 | 1% | 380MHz | 8 | 0% | 448MHz |
| 8x8 | 32 | 5% | 263MHz | 32 | 1% | 443MHz |
| 12x12 | 72 | 11% | 250MHz | 72 | 2% | 375MHz |
| 16x16 | 128 | 20% | 214MHz | 129 | 5% | 375MHz |
| 20x20 | 200 | 32% | 200MHz | 201 | 8% | 319MHz |
| 24x24 | - | - | - | 289 | 12% | 312MHz |

**Table 5: FPGA resource usage (LUT utilization, #BRAMs) and post-synthesis clock frequency.**

# 6. RELATED WORK

There has been a vast body of work in the area of network modeling. The recent emergence of NoCs and increased demand for full-system simulation has posed new constraints and introduced new opportunities that even further expanded this research area; approaches range all the way from analytical modeling to hardware prototyping.

In the area of NoC analytical modeling there has been a significant amount of work examining generalized models that capture a variety of different NoC router architectures [18, 3, 9] as well as analytical models that target specific NoC instances [16]. Compared to FIST, the static nature of analytical modeling does not make it suitable for use within execution-driven simulations. However, such models can be used to construct the load-delay curves used by FIST.

In the context of abstract network modeling for full-system simulations, prior work has looked at the performance-accuracy trade-off for pure software approaches. In [4], the accuracy and performance of five simple network models within a full-system simulator is studied. The "approximate" model shares ideas with FIST, such as the parallel processing of a packet by all affected routers; however, contrary to FIST, packets are still routed through the network. The concept of load-delay curve representation has also been used in the past [15], but in a more complex manner and for entire segments of a high-speed cluster interconnect.

Finally there have been a number of previous attempts at using FPGAs for NoC modeling, such as [25, 27, 26]. However, compared to FIST, these approaches offer cycle-accurate fidelity at the cost of very limited scalability combined with high complexity. Previous work has looked at time-multiplexing multiple simulated routers on a single

virtualized FPGA-resident router [28]. While such an approach can overcome FPGA capacity limitations, it still suffers from implementation complexity, both for building the cycle-accurate router model as well as for handling time-multiplexing.

# 7. CONCLUSION

We presented FIST, a fast, lightweight FPGA-friendly approach to packet latency estimation for full-system simulations. We evaluated two FIST-based models against multi-programmed and multithreaded workloads on 4x4, 8x8 and 16x16 mesh configurations. Our simpler model that relies only on offline training can approximate latency and aggregate throughput values within an error of 6% and 2% respectively and consistently provides simulation time speedups ranging up to 43x for 16x16 mesh networks. We also present a hybrid model that employs online training, by periodically running a cycle-accurate NoC simulator on the side. This hybrid model reduces both latency and aggregate throughput error to less than 2% and can still achieve a simulation time speedup up to 18x for 16x16 mesh networks. Finally, we present results for an FPGA-based hardware implementation of FIST that can simulate up to 24x24 mesh configurations and can achieve 3 to 4 orders of magnitude speedup over software-based cycle-accurate NoC models.

# 8. ACKNOWLEDGMENTS

Funding for this work has been provided by NSF CCF-0811702. We thank the anonymous reviewers and members of the Computer Architecture Lab at Carnegie Mellon (CALCM) for their comments and feedback. We thank Xilinx for their FPGA and tool donations. We thank Bluespec for their tool donations and support.

# 9. REFERENCES

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. PACT, 2008.

[2] Bluespec, Inc. [Online]. http://www.bluespec.com/products/bsc.htm.

[3] Y. M. Boura and C. R.Das. Modeling Virtual Channel Flow Control in Hypercubes. HPCA, 1995.

[4] D. C. Burger and D. A. Wood. Accuracy vs. Performance in Parallel Simulation of Interconnection Networks, 1995.

[5] C. K. Luk.R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, et al. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. PLDI, 2005.

[6] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators, 2007.

[7] E. S. Chung, M. K. Papamichael, E. Nurvitadhi, J. C. Hoe, and K. Mai. ProtoFlex: Towards Scalable, Full System Multiprocessor Simulations Using FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, 2009.

[8] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[9] S. Foroutan, Y. Thonnart, R. Hersemeule, and A. Jerraya. An Analytical Method for Evaluating Network-on-Chip Performance. In *DATE*, 2010.

[10] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, et al. A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS. ISSCC, 2010.

[11] N. E. Jerger, L. S. Peh, and M. Lipasti. Circuit-Switched Coherence. NOCS, 2008.

[12] C. Kim, D. Burger, and S. W. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches. ASPLOS, 2002.

[13] J. Kim, J. Balfour, and W. Dally. Flattened Butterfly Topology for On-Chip Networks. MICRO, 2007.

[14] D. Lenoski and J. Laudon. The SGI Origin: A ccNUMA Highly Scalable Server, 1997.

[15] D. Lugones, D. Franco, D. Rexachs, J. Moure, E. Luque, E. Argollo, A. Falcon, D. Ortega, and P. Faraboschi. High-speed network modeling for full system simulation. IISWC, 2009.

[16] M. Moadeli, A. Shahrabi, W. Vanderbauwhede, and P. Maji. An Analytical Performance Model for the Spidergon NoC with Virtual Channels. AINA, 2010.

[17] T. Moscibroda and O. Mutlu. A Case for Bufferless Routing in On-Chip Networks. ISCA, 2009.

[18] U. Y. Ogras and R. Marculescu. Analytical Router Modeling for Networks-on-Chip Performance Analysis. DATE, 2007.

[19] Open Source Network-on-Chip Router RTL. [Online]. http://nocs.stanford.edu/router/html.

[20] H. Patil, R. Cohn, M. Charney, R. Kapoor, and A. Sun. Pinpointing Representative Portions of Large Intel Itanium Programs with Dynamic Instrumentation. MICRO, 2004.

[21] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, et al. *TILE64 - Processor: A 64-Core SoC with Mesh Interconnect*. ISSCC, 2008.

[22] S. C. Woo, O. Moriyoshi, T. Evan, et al. The SPLASH-2 Programs: Characterization and Methodological Considerations. ISCA, 1995.

[23] A. Singh, W. J. Dally, A. K. Gupta, and B. Towles. GOAL: A Load-Balanced Adaptive Routing Algorithm for Torus Networks, 2003.

[24] Standard Performance Evaluation Corporation. http://www.spec.org/cpu2006.

[25] L. Thuan and M. Khalid. NoC Prototyping on FPGAs: A Case Study Using an Image Processing Benchmark. IEEE EIT, 2009.

[26] U. Y. Ogras, R. Marculescu, H. G. Lee, P. Choudhary, D. Marculescu, et al. Challenges and Promising Results in NoC Prototyping Using FPGAs. *IEEE Micro Special Issue on Interconnects for Multi-Core Chips*, September 2007.

[27] D. Wang, N. E. Jerger, and J. G. Steffan. DART: Fast and Flexible NoC Simulation using FPGAs. WARP, 2010.

[28] P. Wolkotte, P. Holzenspies, and G. Smit. Fast, Accurate and Detailed NoC Simulations. NOCS, 2007.

[29] H. S. W. Xinping, H. S. Wang, X. Zhu, L. S. Peh, and S. Malik. Orion: A Power-Performance Simulator for Interconnection Networks. MICRO, 2002.