Crossing Boundaries in TimeTrial: Monitoring Communications Across Architecturally Diverse Computing Platforms

Joseph M. Lancaster
Joseph G. Wingbermuehle
Jonathan C. Beard
Roger D. Chamberlain

Joseph M. Lancaster, Joseph G. Wingbermuehle, Jonathan C. Beard, and Roger D. Chamberlain, “Crossing Boundaries in TimeTrial: Monitoring Communications Across Architecturally Diverse Computing Platforms,” in *Proc. of Ninth IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, October 2011, pp. 280-287.

Dept. of Computer Science and Engineering
Washington University in St. Louis

2011 Ninth IEEE/IFIP International Conference on Embedded and Ubiquitous Computing

Crossing Boundaries in TimeTrial: Monitoring Communications Across Architecturally Diverse Computing Platforms

Joseph M. Lancaster, Joseph G. Wingbermuehle, Jonathan C. Beard, and Roger D. Chamberlain
Dept. of Computer Science and Engineering, Washington University in St. Louis
{lancaster,wingbej,jbeard,roger}@wustl.edu

***Abstract*—TimeTrial is a low-impact performance monitor that supports streaming data applications deployed on a variety of architecturally diverse computational platforms, including multicore processors and field-programmable gate arrays. Communication between resources in architecturally diverse systems is frequently a limitation to overall application performance. Understanding these bottlenecks is crucial to understanding overall application performance. Direct measurement of inter-resource communications channel occupancy is not readily achievable without significantly impacting performance of the application itself. Here, we present TimeTrial's approach to monitoring those queues that cross platform boundaries. Since the approach includes a combination of direct measurement and modeling, we also describe circumstances under which the model can be shown to be inappropriate. Examples with several micro-benchmark applications (for which the true measurement is known) and an application that uses Monte Carlo techniques to solve Laplace's equation are used for illustrative purposes.**

I. Introduction

TimeTrial is a performance monitoring system optimized for streaming data applications deployed on architecturally diverse platforms [13]. TimeTrial augments the Auto-Pipe application development environment [4], [6] which supports both embedded [21] and high-performance [3] computing across co-designed, architecturally diverse platforms including multicore processors and field-programmable gate arrays (FPGAs).

Streaming data applications are characterized by discrete compute kernels (or blocks) that communicate via explicitly declared channels. Figure 1 shows an example of a signal-processing streaming application with three major stages. In stage 1, a sensor (e.g., a gamma ray telescope [22]) produces data that is split up and sent to parallel processing pipelines in stage 2. Each pipeline performs some functions on the data, which are merged in stage 3. Finally, the results are all compared and stored to disk. Tyson et al. [21] describe the implementation of this application using FPGAs to execute the computationally expensive stage 2 and multicore processors for stages 1 and 3. Languages that directly support streaming applications include Brook [2], StreamIt [20], Streams-C [8], and X/Auto-Pipe [6].

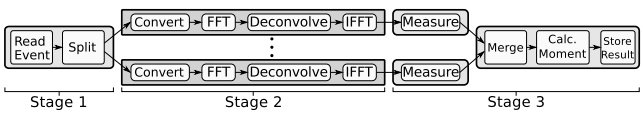

Fig. 1. An example embedded, streaming application from computational astrophysics [21].

The TimeTrial performance monitor instruments streaming data applications that are deployed on a combination of multicore processors and FPGAs. Performance queries (specified using the TimeTrial language [14]) are translated into instrumentation infrastructure that is deployed as part of the Auto-Pipe runtime system. Examples of performance queries might include:

- What is the mean throughput rate from the `Convert` block to the `FFT` block?
- What is the queue occupancy of the queue deployed between the `FFT` block and and the `Deconvolve` block?

Note that the Auto-Pipe runtime system deploys a FIFO queue associated with each communication channel.

While the answer to the first question above helps inform the developer of the pipeline's overall throughput (measured at a given point in the pipeline), it is answers to queries like the second question that help determine which block is the throughput bottleneck. If the queue upstream of `Deconvolve` is mostly full and the queue downstream is almost always empty, we can conclude with some confidence that `Deconvolve` is a rate-limiting block in the pipeline.

TimeTrial supports the notion of performance "metrics" that are directly monitored (e.g., throughput rate, queue occupancy) and a rich set of aggregation functions (e.g., min, max, mean, histogram) that enable aggressive data reduction on the monitored metrics. The aggregations are computed over user-specified intervals, called *frames*, that preserve time-dependent behavior at the time resolution of the frame period.

The automated implementation of the instrumentation to measure metrics and the logic to perform aggregation are described in [14] for communication channels that are within a single compute resource (e.g., the queue upstream of `Deconvolve`, which is deployed on an FPGA). Problems arise when the communication channel crosses platform boundaries, such as the link from the `IFFT` block to the `Measure` block. In this case, a portion of the queue is on the FPGA, a portion of the queue is comprised of buffers associated with whatever low-level communication mechanism(s) are used to move data between the FPGA and the processor (e.g., a PCIe bus or something similar), and a portion of the queue is in the processor's memory. We call a queue that crosses platform boundaries a *virtual queue*.

Measuring the occupancy of this virtual queue is not simply a matter of instrumenting the enqueue and dequeue operations. To perform the aggregation function(s), both enqueue and dequeue *events* must be known on a common platform, which

978-0-7695-4552-3/11 $26.00 © 2011 IEEE
DOI 10.1109/EUC.2011.26

necessitates moving either the enqueue events from the FPGA to the processor or the dequeue events from the processor to the FPGA. Neither of these options is attractive, since a large number of events implies a large data volume (of performance meta-data) must share the processor-FPGA interconnect (such as the PCIe bus example above).

In this paper, we describe TimeTrial's approach to dealing with virtual queues, queues that cross platform boundaries, while preserving the low-impact nature of the performance monitoring that is characteristic of TimeTrial. Example measurements of virtual queues will be compared to ground truth (precise knowledge of the quantity being measured) collected from the micro-benchmark application itself. In addition, we discuss the circumstances where the approach described here is inappropriate, and how users of TimeTrial might come to understand when that is the case. Finally, we apply the approach to measurements of an application that solves Laplace's equation using Monte Carlo methods.

II. Monitoring Virtual Queues

The direct measurement of communication channels (and their associated queues) that cross platform boundaries is incompatible with the notion of low-impact monitoring. While some metrics of interest, such as throughput rate, can be effectively measured at one end of the channel or the other, other metrics, such as queue occupancy, require information from both the head and the tail of the queue.

TimeTrial's approach to virtual queue monitoring is to instrument what it can and use a performance model of the underlying system to infer what it cannot directly measure, estimating the information that is missing. As such, it is important not only to provide the performance quantities that were estimated, but also to provide guidance as to the quality of the estimates. In what follows, we will focus on querying the occupancy of virtual queues. It is anticipated that the same techniques will be similarly effective for other metrics that require detailed event information across platform boundaries (e.g., latency through virtual queues); however, this is left for confirmation in future work. It is clearly true for some aggregated metrics such as mean latency, which is directly related to mean occupancy via Little's Law.

As defined here, virtual queues are comprised of several constituent components, all chained together to comprise the communications channel between two compute blocks. For a channel that moves data between a multicore processor and an FPGA, the components will include buffers in user space on the processor, kernel buffers on the processor, a physical bus that transfers the data to the FPGA (e.g., PCIe), buffers in the DMA engine on the FPGA card, and application buffers deployed (by the Auto-Pipe runtime system) on the FPGA. While many of these constituent components of the communications channel are opaque (i.e., they cannot be directly monitored by TimeTrial), the two components at either end of the chain (comprising the head of the queue and the tail of the queue) are visible to TimeTrial and can therefore be monitored. Whenever a user requests the occupancy of a virtual queue, TimeTrial deploys monitors for the head and tail sub-queues for which it has visibility.

A. Approach to Virtual Queue Occupancy

To measure the occupancy of a queue over time, one can record a series of inter-insertion and inter-departure time stamps at the head and tail of a queue, respectively. The queue can then be "replayed" using a simple simulation based on this event trace. Practically, this approach is useful for queues that have small volumes of data moving through them. However, the amount of trace data that needs to be stored quickly becomes burdensome as the data volume approaches TimeTrial's ability to store the event trace. This problem becomes particularly acute on FPGA platforms, since there is very limited buffering on the FPGA to store events.

For regular queues (i.e., those that reside entirely on a single compute resource), TimeTrial monitors the insertion and removal events and aggregates these to a histogram (or some other requested function) of queue occupancy on-line. For virtual queues, access to the insertion or removal event trace is impractical since they must be communicated across platform boundaries. This would likely overwhelm the communication bus between these resources for many applications causing unwanted performance interference.

Instead, TimeTrial recreates the queue occupancy through an empirically-driven, stochastic, discrete-event simulation. TimeTrial measures the time between each insertion into the queue and between each removal from the queue. Both streams of delta time values are then aggregated on-line into an inter-insertion time histogram and an inter-departure time histogram. These histograms are then used to model an arrival process and a departure process from the virtual queue. By sampling from the histograms, a stochastic discrete-event simulation is performed which samples from these distributions to "replay" the virtual queues over time.

TimeTrial performs aggregations over a frame. This produces one set of inter-transfer time histograms per frame. Having more than one histogram allows for the stochastic simulation to be non-stationary, that is, the distributions change over time. Hence, the measurements reflect changes in application performance over the course of its execution.

There are two modeling assumptions that are present in the stochastic simulation that warrant consideration. First, when replaying the virtual queue dynamics the stochastic simulator is assuming that the virtual queue acts as an ideal queue (e.g., insertions at some time t are immediately available for removal at time t). This assumption presumes there is no inherent latency present in the underlying implementation of the communications channel.

Second, the stochastic simulation makes the assumption that the insertion and removal processes are independent and identically distributed (iid). Significant auto-correlation in either process, and/or cross-correlation between the processes, constitute another potential source for error.

If one or more of these modeling assumptions is untrue for the application being executed, there is a reasonable chance

that the stochastic simulation predictions for the occupancy of the virtual queue will diverge from the measured occupancy of the sub-queues at the head and the tail of the virtual queue. This suggests an approach for performing a sanity check on the model's predictions. If the predicted queue occupancies do not align with the measured occupancies of the visible sub-queues, the model is not to be trusted. Note that if the model predictions and the measured sub-queue results do match, that is no proof that the model predictions are correct. However, when there is not a match, it is clear that the model's predictions cannot be trusted.

In summary, the procedure for responding to a query that asks for the occupancy of a virtual queue is as follows:

1) Instrument the ingress point (tail) of the queue and its associated sub-queue (that is visible to TimeTrial) on the source computational resource and the egress point (head) of the queue and its associated sub-queue (that is visible) on the destination computational resource.
2) Execute the application, recording a histogram of the inter-insertion times at the tail of the queue and a histogram of the inter-departure times at the head of the queue. Also record occupancy histograms of the sub-queues at the head and tail.
3) Using the inter-insertion time and inter-departure time histograms to drive the pseudo-random number generator distribution, perform a stochastic discrete-event simulation, predicting the occupancy of the virtual queue.
4) Compare the stochastic simulation predictions with the measured sub-queue occupancy distributions.
5) If the comparison of step 4 succeeds, report the occupancy of the virtual queue to the user.
6) If the comparison of step 4 fails, one or more of the stochastic simulation model assumptions are not true for the application, and the stochastic simulation's predictions should not be trusted.

B. Assessment

We assess the above described approach to understanding the dynamics of a virtual queue through the use of several micro-benchmark applications. These applications are designed to either: (1) have a known queue occupancy by construction; or (2) effectively record the queue occupancy during execution. Either way, we know ground truth and can therefore effectively assess the model.

The family of micro-benchmarks that are used are designed to emulate the queue activity in a traditional queue-server combination (i.e., queueing station) from queueing theory [11]. Figure 2 illustrates a single-stage queueing station that includes an arrival process from the left (that delivers "jobs" into the system with mean arrival rate λ), a FIFO queue, and a service process on the right (that services "jobs" from the queue with mean service rate μ).

For our purposes, a "job" will be represented by a single data element (8 bytes in size), and the distribution of the arrival process and the service process will either be Markovian (i.e., exponential inter-arrival and/or service times) or deterministic (i.e., constant inter-arrival and/or service times).

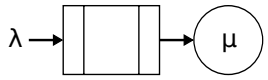

Fig. 2. Single-stage queueing station.

The first micro-benchmark emulates the queue activity in a classic $M/M/1$ queueing station [11]. In this notation, the first M denotes a Markovian arrival process, the second M denotes a Markovian service process, and the 1 denotes a single server. Figure 3 shows the Auto-Pipe micro-benchmark application, where the `source` block is acting as the arrival process (with pseudo-random numbers drawn from an exponential distribution, mean $1/\lambda$, provided by the left-most `PRNG` block). The `server` block is acting as the server (with pseudo-random numbers drawn from an exponential distribution, mean $1/\mu$, provided by the second `PRNG` block). The queue shown on the edge between `source` and `server` is the queue of Figure 2. Note that there are queues on the other application edges as well, they are simply not drawn in the figure.

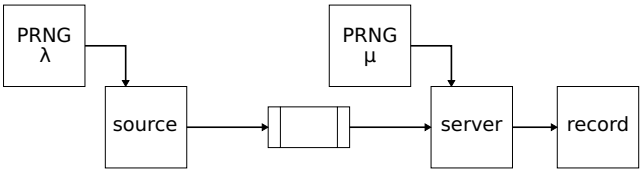

Fig. 3. Micro-benchmark application that mimics queue activity of the single-stage queueing station.

To enable playback of true queue occupancy, the data value delivered from `source` to `server` contains the inter-insertion time between two insertion events on the queue. The data values delivered from the `server` block to the `record` block include both the above inter-insertion time and the inter-departure time experienced at the `server` block. This stream of data enables us to recreate the actual dynamics of the queue via a trace-driven simulation after the run has completed. This trace-driven simulation provides our best understanding of ground truth (i.e., what really happened in the queue).

Due to potential confusion that might result from our use of two different simulations, each for a distinct purpose, we will use the following terms in the discussion below.

- *Stochastic simulation* refers to the simulation model introduced in Section II-A that forms the basis for TimeTrial's predictions of queue occupancy. This simulation will be utilized each time a user requests TimeTrial to measure the occupancy of a virtual queue.
- *Trace-driven simulation* refers to the simulation introduced in the previous paragraph that uses trace information recorded by a micro-benchmark application which represents ground truth for the inter-insertion and inter-departure times of the virtual queue. This simulation is only available if the application records these time traces.

The $M/M/1$ micro-benchmark was deployed on 2 processors (the `source` and its random number generator on one processor, the `server` and its random number generator on the other) and the queue in question was instrumented using TimeTrial, asking for a histogram of the queue occupancy. The utilization of the queueing server for this run, $\rho = \lambda/\mu$ was set to 0.9. Figure 4 shows two versions of the queue's occupancy in this micro-benchmark execution: (1) the trace-driven simulation result from the micro-benchmark output; and (2) the measured value from TimeTrial. The obvious alignment of these results simply indicates that the micro-benchmark is doing what we expect it to do and TimeTrial is readily capable of instrumenting queues that reside entirely on one compute resource (in this case a multicore processor chip).

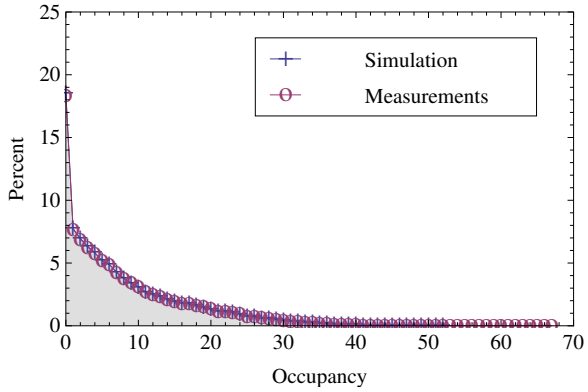

Fig. 4. Queue occupancy in $M/M/1$ micro-benchmark deployed in software.

The above micro-benchmark was redeployed on an FPGA (all blocks except `record`) and the experiment repeated (this time with $\rho = \lambda/\mu = 0.8$) giving the results in Figure 5. Again, the trace-driven simulation results and the measured results from TimeTrial are in close agreement.

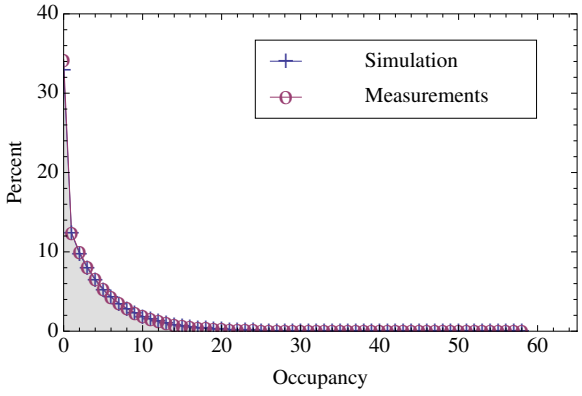

Fig. 5. Queue occupancy in $M/M/1$ micro-benchmark deployed in hardware.

Next we will illustrate the approach using a mapping that includes a virtual queue. The `server` block is mapped to an FPGA and the remaining blocks are all mapped to processor cores. Figure 6 shows the queue occupancy of the virtual queue that crosses the processor-to-FPGA boundary based upon the trace-driven simulation that represents ground truth. Figure 7 shows TimeTrial's estimate of the queue occupancy based on the stochastic simulation described above. It is obvious by comparison with Figure 6 that the actual queue occupancy is significantly different from TimeTrial's estimate. Clearly, TimeTrial needs a better model.

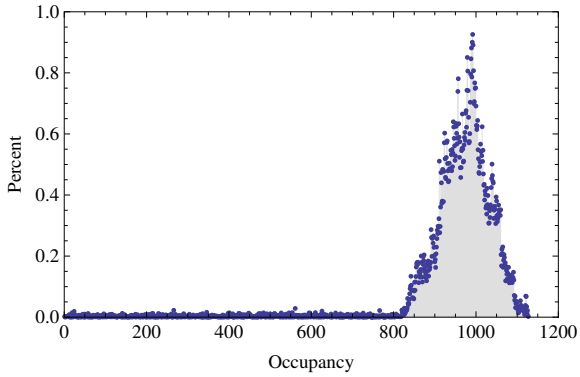

Fig. 6. True queue occupancy in $M/M/1$ micro-benchmark with software-to-hardware virtual queue.

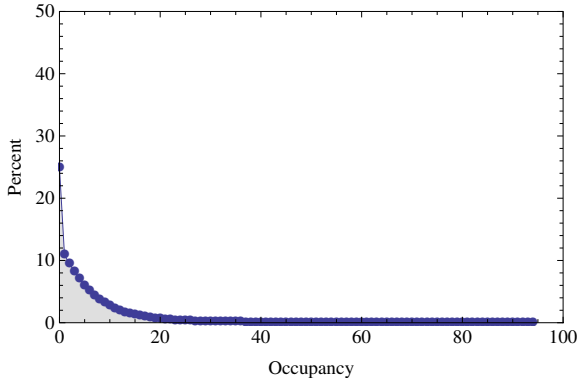

Fig. 7. Modeled queue occupancy in $M/M/1$ micro-benchmark with software-to-hardware virtual queue.

III. Interconnect Bus Model

Guiding the consideration of a more robust model are the two modeling assumptions described in Section II-A: (1) the existence of an ideal queue, and (2) iid insertion and removal processes. Since the chosen micro-benchmark matches the second assumption (i.e., its inter-arrival time distribution and its service distribution are both iid), we next investigate a more robust model of the underlying communications channel that implements the virtual queue.

The physical system that we are using consists of a two-socket motherboard that contains a pair of AMD multicore chips and a Xilinx Virtex-4 FPGA on a PCI-X bus. If we treat the bus as a "server," in the sense of a queueing model, we can extend the model of the communications channel into the

two-stage tandem queueing network illustrated in Figure 8. Here, the first server (with constant service rate μ_1 and a deterministic service time distribution) models the bus and the second server (with mean service rate μ_2) models the downstream compute block. The stochastic simulation can then "replay" the activity in the virtual queue by simulating the tandem queueing network.

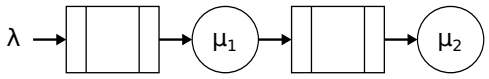

Fig. 8. Two-stage tandem queue.

In this system, data to be moved across the PCI-X bus is buffered (by the low-level system software) until a minimum bulk transfer size is reached, at which point a bulk DMA transfer moves the data across the bus. This property of the communication channel can be readily reflected in the tandem queueing network model by ensuring that the first server (representing the PCI-X bus) performs bulk transfers. In the stochastic simulation model, it allows data to buffer in the upstream queue until `BulkSize` data elements are queued, at which point it "services" them all together, removing them from the upstream queue and inserting them into the downstream queue. In this model, the instantaneous occupancy of the virtual queue consists of the occupancy of the upstream queue of the bus server plus the occupancy of the downstream queue of the bus server plus the number in service in the bus.

Returning to the $M/M/1$ micro-benchmark whose true virtual queue occupancy histogram is shown in Figure 6, the queue occupancy histogram that TimeTrial predicts using the stochastic simulation based on the tandem-queue model is shown in Figure 9. These two figures are in reasonable agreement with one another. In addition, the sub-queue histograms directly measured by TimeTrial of the head and tail of the virtual queue (not shown) are also in agreement with Figures 6 and 9.

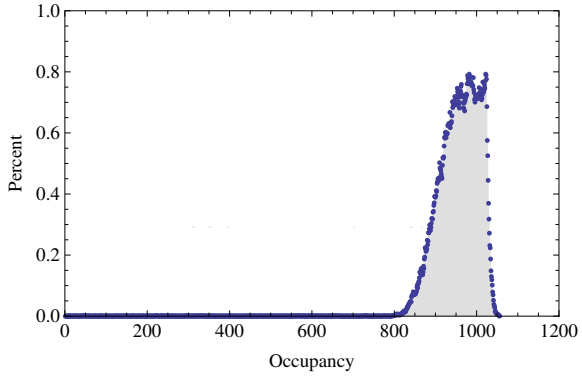

Fig. 9. Modeled queue occupancy in $M/M/1$ micro-benchmark with software-to-hardware virtual queue, using tandem queue model.

We next assess the appropriateness of the tandem-queue model on a few additional micro-benchmarks. Figure 10 shows the results of a trace-driven simulation giving true queue occupancy for an $M/M/1$ micro-benchmark with the mapping reversed to give a hardware-to-software virtual queue. (Here, `source` and its associated random number generator are mapped to the FPGA.) Figure 11 shows the results of the stochastic simulation using the tandem-queue model. As with the software-to-hardware virtual queue, there is reasonable agreement between the modeled queue occupancy and the true queue occupancy.

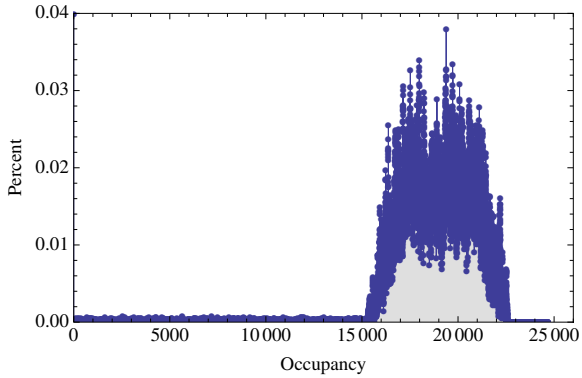

Fig. 10. True queue occupancy in $M/M/1$ micro-benchmark with hardware-to-software virtual queue.

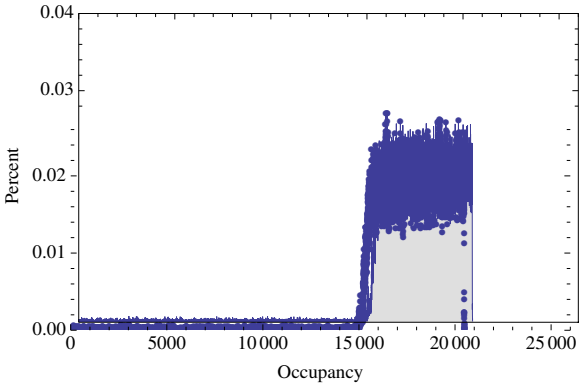

Fig. 11. Modeled queue occupancy in $M/M/1$ micro-benchmark with hardware-to-software virtual queue.

Next we explore the sensitivity of the model to the distribution of the service process. In this micro-benchmark, the `server` block is set to perform dequeue operations at a constant rate. This micro-benchmark corresponds to an $M/D/1$ queueing station, where the arrival process is Markovian and the service process is deterministic. Figure 12 presents the trace-driven simulation results that indicate true queue occupancy for a software-to-hardware virtual queue for this micro-benchmark (i.e., `source` is mapped to a processor core and `server` is mapped to an FPGA). Figure 13 presents the stochastic simulation model of the same micro-benchmark.

As is clear in the figures, the shape of the virtual queue occupancy is well represented in the stochastic simulation model. As with the previous micro-benchmarks, the occupancy

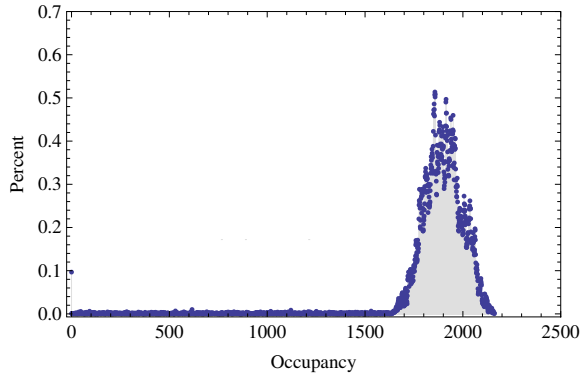

Fig. 12. True queue occupancy in $M/D/1$ micro-benchmark with software-to-hardware virtual queue.

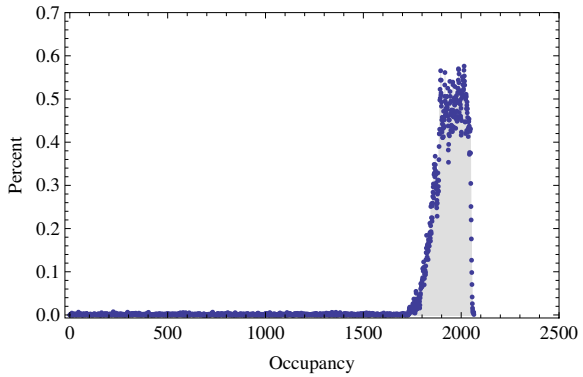

Fig. 13. Modeled queue occupancy in $M/D/1$ micro-benchmark with software-to-hardware virtual queue.

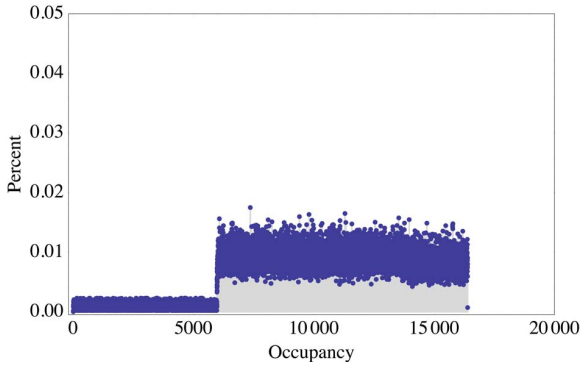

Fig. 14. Modeled queue occupancy in correlated micro-benchmark with hardware-to-software virtual queue.

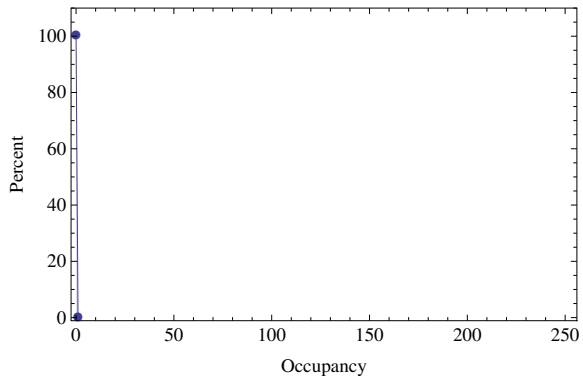

Fig. 15. Hardware sub-queue occupancy in correlated micro-benchmark with hardware-to-software virtual queue.

is strongly influenced by the bulk transfer characteristics of the bus, so much so that it does not even qualitatively resemble the theoretically expected queue occupancy for an $M/D/1$ queueing station at all.

In each of the above micro-benchmark examples, the stochastic simulation model does a reasonably good job of indicating the general properties of the occupancy of the virtual queue. However, there are instances where the stochastic simulation model will fail to accurately represent the true occupancy of the virtual queue. An example of this is illustrated in Figures 14 to 16, which show the predicted queue occupancy from the stochastic simulation (Figure 14), the measured sub-queue occupancy from the hardware side (Figure 15), and the measured sub-queue occupancy from the software side (Figure 16) for a micro-benchmark with a hardware-to-software virtual queue. In this case, the source random number stream had significant auto-correlation (i.e., the source was not iid).

It is clear from the figures that the sub-queue occupancies measured at the head and the tail of the virtual queue do not agree with the overall queue occupancy predicted by the stochastic simulation model. While for this particular instance the reason for the lack of agreement is well understood (a non-iid source), in general the use of the head and tail sub-queue occupancies serves as a check on the appropriateness of the stochastic simulation model, but does not necessarily give a reason when there isn't agreement.

IV. Solution to Laplace's Equation

Laplace's equation is a second-order partial differential equation (PDE) [19] that has several uses, including modeling stationary diffusion (such as heat) and Brownian motion. For heat, given the temperature at the boundaries of an object, solutions to Laplace's equation provide the interior temperatures at equilibrium.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

The ease of solving Laplace's equation depends on the nature of the boundary conditions. An analytic solution exists for simple boundary conditions, however, for many boundary conditions, no analytic solution exists and numerical solutions are needed [5]. There are several approaches for numerical solutions. One approach is Gauss-Seidel iterations [19]. This method converges quickly, but it requires that the complete grid be stored in memory. Another method is Monte Carlo simulation. This technique is provably correct [17], but converges slowly. Nevertheless, this method is useful if a small number of interior points are needed. This is because the Monte Carlo

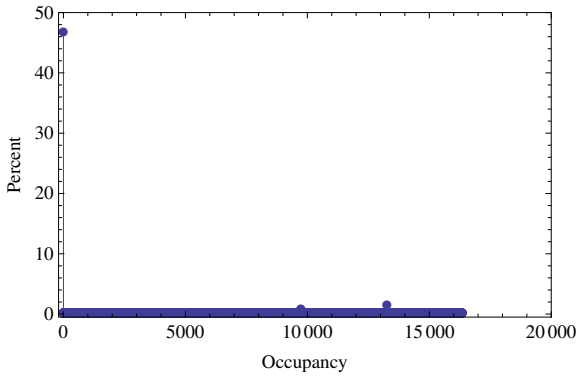

Fig. 16. Software sub-queue occupancy in correlated micro-benchmark with hardware-to-software virtual queue.

method does not require storing the entire grid, since the grid is implicit, and only those points that are of interest need be computed.

Figure 17 shows the topology of an Auto-Pipe implementation of a Monte Carlo solver for Laplace's equation. The pseudo-random numbers from the `PRNG` block are distributed to 8 independent blocks performing random `Walk`s executing the Monte Carlo solution [17]. The results are combined in the `Avg` blocks and are recorded in the `Print` block.

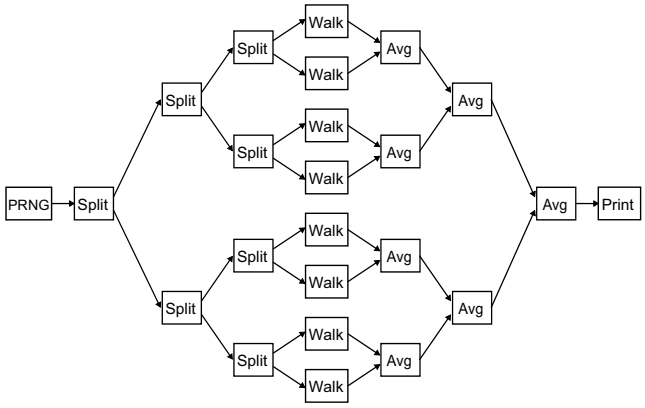

Fig. 17. Topology of Auto-Pipe application that solves Laplace's equation using Monte Carlo methods.

We use this application to illustrate the ability of TimeTrial to monitor temporal properties of an application. We set TimeTrial's frame period to 1 second (meaning that TimeTrial aggregates performance meta-data over 1 second intervals and reports performance once per second) and asked for the mean queue occupancy for the queue from `PRNG` to the first `Split` and also for the queue from the last `Avg` to `Print`. The application runs for 60 seconds. For this execution, all of the blocks except `Print` have been assigned to the FPGA and `Print` has been assigned to a processor core. As a result, the first monitored queue is entirely on the FPGA and the second monitored queue is a virtual queue, moving data from the FPGA to the processor core.

Figure 18 shows the mean queue occupancy over time for the queue immediately downstream of the `PRNG` block. The capacity of this queue is 128, and as is readily apparent from the graph, within the first second of execution this queue is full and stays full for the duration of the run.

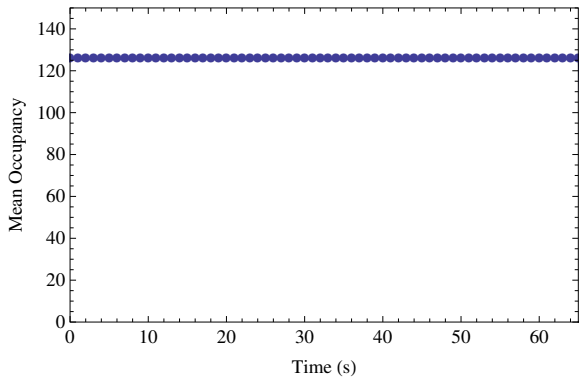

Fig. 18. Mean queue occupancy out of `PRNG` block in Monte Carlo solution to Laplace's equation. This is a hardware-to-hardware queue, entirely within the FPGA.

Figure 19 shows the mean queue occupancy over time for the queue upstream of the `Print` block. Here, the effect of bulk transfers across the bus is clearly apparent. Data destined for the `Print` block is buffered, waiting to be transferred across the PCI-X bus, until the very end of the application's execution, at which point the buffers are flushed and all of the results are delivered to the `Print` block. Note that if the latency of individual data elements moving from `Avg` to `Print` is an important performance metric for this application, simply monitoring the average rate across the edge is not sufficient to discover the true nature of this virtual queue's activity.

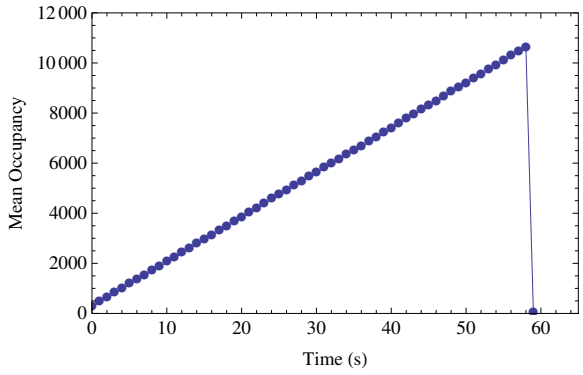

Fig. 19. Mean queue occupancy into `Print` block in Monte Carlo solution to Laplace's equation. This is a hardware-to-software virtual queue, moving data from the FPGA to a processor core.

Further monitoring using TimeTrial (not shown here) demonstrates that the `Walk` blocks are the current performance bottleneck, and increasing the parallelism to greater than 8 `Walk` blocks is an important step in improving the overall application performance.

V. Related Work

There has been a great deal of work in monitoring the performance of executing applications, from gprof [9] and TAU [18] for platforms using traditional processor cores to the tools of Koehler et al. [12] aimed at FPGA designs. Each of the above systems is tunable with respect to the impact it can have on the performance of the application (i.e., the tradeoff between performance impact and measurement fidelity is controlled by the user). None of these systems, however, effectively measures communications across platform boundaries without incurring significant performance impact.

TimeTrial adopts the approach of dedicating resources to the monitoring task in an attempt to minimize performance impact. This approach has its origins in the work of Maloney and Reed [15], and recent examples include hardware performance monitors for the Cell processor [7], soft-core processors deployed on FPGAs [10], and traditional processors [1].

VI. Conclusions and Future Work

This paper has presented TimeTrial's approach to monitoring virtual queues in a streaming data application, those queues that cross the boundaries of the computing platform executing the application. For those measurements that require data collection at both the head and the tail of the queue, histograms of inter-insertion times and inter-departure times are collected at each end of the queue, and a stochastic discrete-event simulation model is used to recreate the dynamics of the virtual queue in question.

As a partial verification of the appropriateness of the stochastic simulation model, the portions of the virtual queue that are visible to TimeTrial are also monitored, and if the stochastic simulation results are not consistent with the measurements of the sub-queues at the head and the tail of the virtual queue, the stochastic simulation model is deemed incorrect and is discarded. In this way, TimeTrial works to not only model the activity within the virtual queue, but also helps to verify whether or not its internal model is appropriate.

Future work includes the development of a state-space Markovian model of Figure 8's tandem queueing network with a bounded capacity queue between the two servers. This has the potential to provide an analytic solution, which might obviate the need to perform stochastic simulation. Even if a closed-form solution to the analytic model is unavailable, a numeric solution can be less expensive than a stochastic discrete-event simulation run. Alternatively, approximate solution methods for the bounded capacity queue with upstream blocking do exist (see, e.g., [16]), which can also be less expensive to compute than simulation.

Acknowledgements

This research was supported by the National Science Foundation through grants CNS-0751212, CNS-0905368, and CNS-0931693.

References

[1] S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci, "A scalable cross-platform infrastructure for application performance tuning using hardware counters," in *Proc. of ACM/IEEE Supercomputing Conf.*, 2000.

[2] I. Buck, T. Foley, D. Horn, J. Sugerman, and K. Fatahalian, "Brook for GPUs: Stream computing on graphics hardware," *ACM Trans. on Graphics*, vol. 23, no. 3, pp. 777–786, Aug. 2004.

[3] R. D. Chamberlain, J. Buhler, M. A. Franklin, and J. H. Buckley, "Application-guided tool development for architecturally diverse computation," in *Proc. of ACM Symp. on Applied Computing*, Mar. 2010, pp. 496–501.

[4] R. D. Chamberlain, M. A. Franklin, E. J. Tyson, J. H. Buckley, J. Buhler, G. Galloway, S. Gayen, M. Hall, E. B. Shands, and N. Singla, "Auto-Pipe: Streaming applications on architecturally diverse systems," *Computer*, vol. 43, no. 3, pp. 42–49, Mar. 2010.

[5] S. J. Farlow, *Partial Differential Equations for Scientists and Engineers*. Dover Publications, 1993.

[6] M. A. Franklin, E. J. Tyson, J. Buckley, P. Crowley, and J. Maschmeyer, "Auto-pipe and the X language: A pipeline design tool and description language," in *Proc. of Int'l Parallel and Distributed Processing Symp.*, Apr. 2006.

[7] M. Genden, R. Raghavan, M. Riley, J. Spannaus, and T. Chen, "Real-time performance monitoring and debug features of the first generation Cell processor," in *Proc. of 1st Workshop on Tools and Compilers for Hardware Acceleration*, Sep. 2006.

[8] M. B. Gokhale, J. M. Stone, J. Arnold, and M. Kalinowski, "Stream-oriented FPGA computing in the Streams-C high level language," in *Proc. of IEEE Int'l Symp. on FPGAs for Custom Computing Machines*, 2000, pp. 49–58.

[9] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," in *Proc. of SIGPLAN Symp. on Compiler Construction*, 1982, pp. 120–126.

[10] R. Hough, P. Krishnamurthy, R. D. Chamberlain, R. K. Cytron, J. Lockwood, and J. Fritts, "Empirical performance assessment using soft-core processors on reconfigurable hardware," in *Proc. of Workshop on Experimental Computer Science*, Jun. 2007.

[11] L. Kleinrock, *Queueing Systems, Volume 1: Theory*. John Wiley & Sons, 1975.

[12] S. Koehler, J. Curreri, and A. D. George, "Performance analysis challenges and framework for high-performance reconfigurable computing," *Parallel Computing*, vol. 34, no. 4-5, pp. 217–230, May 2008.

[13] J. M. Lancaster, E. F. B. Shands, J. D. Buhler, and R. D. Chamberlain, "TimeTrial: A low-impact performance profiler for streaming data applications," in *Proc. of IEEE Int'l Conf. on Application-specific Systems, Architectures and Processors*, Sep. 2011.

[14] J. M. Lancaster, J. G. Wingbermuehle, and R. D. Chamberlain, "Asking for performance: Exploiting developer intuition to guide instrumentation with TimeTrial," in *Proc. of 13th IEEE Int'l Conf. on High Performance Computing and Communications*, Sep. 2011.

[15] A. D. Malony and D. A. Reed, "A hardware-based performance monitor for the Intel iPSC/2 hypercube," in *Proc. of 4th Int'l Conf. on Supercomputing*, 1990, pp. 213–226.

[16] H. Perros and T. Altiok, "Approximate analysis of open networks of queues with blocking: Tandem configurations," *IEEE Trans. Soft. Eng.*, vol. 12, pp. 450–461, 1986.

[17] J. F. Reynolds, "A proof of the random-walk method for solving Laplace's equation in 2-D," *The Mathematical Gazette*, vol. 49, no. 370, pp. 416–420, Dec. 1965.

[18] S. S. Shende and A. D. Malony, "The TAU parallel performance system," *Int'l J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 287–311, 2006.

[19] W. A. Strauss, *Partial Differential Equations: An Introduction*. Wiley, 1992.

[20] W. Thies, M. Karczmarek, and S. Amarasinghe, "StreamIt: A language for streaming applications," in *Proc. of 11th Int'l Conf. on Compiler Construction*, 2002, pp. 179–196.

[21] E. J. Tyson, J. Buckley, M. A. Franklin, and R. D. Chamberlain, "Acceleration of atmospheric Cherenkov telescope signal processing to real-time speed with the Auto-Pipe design system," *Nuclear Inst. and Methods in Physics Research A*, vol. 585, no. 2, pp. 474–479, Oct. 2008.

[22] T. Weekes *et al.*, "VERITAS: the very energetic radiation imaging telescope array system," *Astroparticle Physics*, vol. 17, no. 2, pp. 221–243, May 2002.