Crossing Timezones in the TimeTrial Performance Monitor

Joseph M. Lancaster
Roger D. Chamberlain

Joseph M. Lancaster and Roger D. Chamberlain, “Crossing Timezones in the TimeTrial Performance Monitor,” in *Proc. of 2010 Symposium on Application Accelerators in High Performance Computing*, July 2010.

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

Crossing Timezones in the TimeTrial Performance Monitor

Joseph M. Lancaster and Roger D. Chamberlain
Dept. of Computer Science and Engineering
Washington University in St. Louis
lancaster@wustl.edu, roger@wustl.edu

***Abstract*—TimeTrial is a performance monitoring tool designed to enhance the understanding of how and why a streaming data application is performing when deployed on an architecturally diverse computer. A challenge that exists in architecturally diverse systems is that different computing platforms (e.g., processor core, FPGA) have different clocks, and the notion of time as measured on one platform does not directly compare to that measured on another platform. Here, we describe the global time model employed in the TimeTrial performance monitor, and demonstrate measurements made with TimeTrial that require timestamp comparisons across the disparate time domains (called timezones).**

I. Introduction

Given the ever-increasing demand for computational performance coupled with the recent move toward many-core systems at the expense of increase of single-core performance gains, some developers are utilizing architecturally diverse systems as an approach to high performance computing. An architecturally diverse system combines traditional multicore processors with non-traditional components such as FPGAs or GPUs.

While there is significant promise for performance increases for many applications, the development task is noticeably more difficult. An important piece of this difficulty is the lack of comprehensive tool support. TimeTrial is a performance debugging tool designed to assist application developers who are deploying streaming data applications onto architecturally diverse systems [1], [2]. In contrast to performance monitors such as `gprof` [3], which sample the executing application, TimeTrial dedicates system resources to the monitoring task (e.g., FPGA area, processor core), similar to the instrumentation built into the Cell processor [4], and uses those resources to aggressively compress performance data under user control. In this way, TimeTrial aims to have minimal impact on the performance of the monitored application. Another monitoring system has been developed at the Univ. of Florida [5]. It is focused on monitoring arbitrary signals within a VHDL design, while TimeTrial focuses on streaming applications that have been expressed in a variety of languages.

A number of the capabilities of TimeTrial have previously been described in [1], [2]. These results have focused on measuring the occupancies of the input queues for both the software stages, hardware stages, and treating all the queueing between the CPU and the FPGA as well as the FPGA to CPU as *virtual queues*. Virtual queues combine a number of physical FIFOs that exist in the communication system into a single entity. Occupancy measurements of application queues help determine where in the application bottlenecks exist over time and how often particular bottlenecks are affecting the overall application performance. TimeTrial can also measure the duty cycle of signals in the FPGA design. For instance, one useful measurement is the fraction of time that a stage is blocked over a portion of the run. Combined, these two types of measurements provide powerful insight into the dynamic performance characteristics of the application (e.g., the throughput over time).

Performance measurements are periodically sent from the various TimeTrial monitoring agents back to the TimeTrial server running on a host CPU core. Typically, these results are post-processed to further summarize the performance and presented in a GUI form to the developer. However, using the TimeTrial language [2], the developer has explicit control over what to measure, how much the measurements are aggregated and in what form the results are presented. Since TimeTrial agents share system interconnects with the application and FPGAs can have tight memory constraints, TimeTrial agents aggregate traces into the desired statistic (e.g., mean) locally where possible.

When an application is deployed on multiple computing resources, it is frequently the case that time is measured differently on one or more of the resources. A clear example of this is a processor core versus an FPGA. On a processor core, system calls return the current time at nanosecond resolution. On an FPGA, a counter can measure relative time with the resolution of the clock period, which is often application specific. Here, we describe our approach to resolving time across multiple distinct computing resources, each in their own time domain (which we call a timezone), in the TimeTrial performance monitor.

II. Timezones and Virtual Time

While the approach below generalizes to more than two computing resources, for brevity we will constrain the discussion to a particular circumstance containing only two resources, a processor core and an FPGA. Using an appropriate system call, processor time (denoted t_P) is available to executing software with a known resolution. TimeTrial provides the FPGA time (denoted t_F) using a cycle counter in the

reconfigurable logic design with the resolution of the FPGA clock period.

With knowledge of the relative rates of the two available timestamps, t_P and t_F, one can define a global timestamp (which we call virtual time and denote t_V) and relate the various resource timestamps to virtual time via a set of linear transformations, i.e.,

$$t_V = s_P \cdot t_P + b_P \tag{1}$$

and

$$t_V = s_F \cdot t_F + b_F. \tag{2}$$

where s_P and s_F encode the clock rate differences between the two platforms, and b_P and b_F represent the different time offsets. As an example, consider a processor clocked at 1 GHz, an FPGA clock running at 250 MHz (4 ns resolution), and a desired virtual time with 1 ns resolution. In this case, $s_P = 1$, $s_F = 4$, one of the offsets (say, b_P) can be set to an arbitrary value, and we need to discover (experimentally) the last remaining offset, b_F.

The TimeTrial performance monitor estimates the unknown offset b_F by performing a timezone calibration that records sets of three values: $t_{P,1}$, $t_{F,2}$, and $t_{P,3}$. Short messages are sent from the processor to the TimeTrial agent on the FPGA to retrieve the FPGA cycle counter. The value of $t_{P,1}$ is recorded immediately before sending the message by querying the processor cycle counter. Similarly, $t_{P,3}$ is recorded on the receipt of the response from the TimeTrial FPGA agent. The response message contains the FPGA cycle counter, $t_{F,2}$, as its payload. A typical calibration task sends one thousand of these messages and a suitable subset of those with minimum round trip times are used for calibration.

Given experimental values for $t_{P,1}$, $t_{F,2}$, and $t_{P,3}$ and the known values of s_P, s_F we can now reason about the value of b_F. Given causality, we know that

$$t_{V,1} < t_{V,2} < t_{V,3} \tag{3}$$

substituting for t_V,

$$s_P t_{P,1} + b_P < s_F t_{F,2} + b_F < s_P t_{P,3} + b_P \tag{4}$$

yields the following:

$$b_F > s_P t_{P,1} + b_P - s_F t_{F,2} \tag{5}$$

and

$$b_F < s_P t_{P,3} + b_P - s_F t_{F,2}. \tag{6}$$

These are the bounds on the true value of b_F. Setting b_P to zero, this further simplifies to:

$$b_F > s_p t_{P,1} - s_F t_{F,2} \tag{7}$$

and

$$b_F < s_p t_{P,3} - s_F t_{F,2}. \tag{8}$$

It is convenient to assume that the virtual timestamp $t_{V,2}$ is midway between $t_{V,1}$ and $t_{V,3}$ (i.e., this makes the assumption that the communication between the processor and FPGA is symmetric in delay). Under this assumption, $t_{V,2}$ is the midpoint between $t_{V,1}$ and $t_{V,3}$:

$$t_{V,2} = (t_{V,1} + t_{V,3})/2 \tag{9}$$

and b_F can be computed from (2):

$$b_F = t_{V,2} - s_F t_{F,2}. \tag{10}$$

In practice, our knowledge of s_F is not perfect, so a least mean squares curve fit is performed on the set of calibration points to estimate both s_F and b_F.

Given the above, TimeTrial can now convert both processor timestamps and FPGA timestamps into virtual timestamps. This enables reasoning about performance of an application as a whole independent of the resource on which individual events occur.

III. Experiments

To illustrate the benefits of employing a global virtual time within TimeTrial, we instrumented Mercury BLASTN, a computational biology application deployed on multicore processors and an FPGA [6], [7], [8]. Mercury BLASTN is a streaming application that compares two strings, each representing DNA, to find similarities between them. The application is decomposed into a number of pipelined stages, some deployed on the FPGA and others on the processors. Figure 1 shows the deployment of the application.

Mercury BLASTN has several timezones. On the processor portion of the application, there is one timezone per processor core. However, we utilize the cycle-counter synchronization feature of modern Linux kernels which synchronizes the cycle counters across multiple cores and multiple chips. This leaves us with effectively one timezone on the processor portion of Mercury BLASTN. The FPGA portion of Mercury BLASTN is currently configured so that all the stages run in the same clock domain. When Mercury BLASTN is configured with multiple clock domains, TimeTrial provides one cycle counter per domain. In our configuration, however, this leaves us with two distinct timezones. When data is transferred between timezones, TimeTrial must employ its notion of virtual time to normalize the performance results.

In Mercury BLASTN, the virtual queues to and from the FPGA previously had unknown latency characteristics for data flowing through the application. The two virtual queues are shown in Figure 1 as single queues going to and from the FPGA. Using TimeTrial, four different latencies of interest were measured for each pass of the database. First, the latency for sending the first DNA base from the processor to the FPGA was recorded. Referring to Figure 1, this corresponds to the latency of the first datum from T1 to T2. One would expect this to be the minimum latency scenario since the queueing delay should be minimized due to empty queues. The latency from T1 to T2 of the last DNA base in the database stream from processor to FPGA was also measured. In a similar manner, the latency of the first result returned from the FPGA to the processor (i.e., from T3 to T4) was also measured. Finally, the latency of the end of stream marker from T3 to T4 was

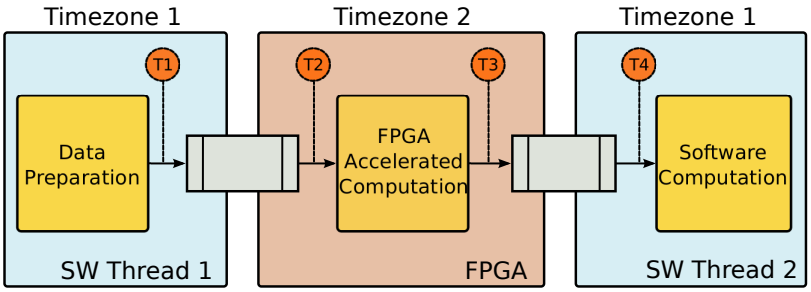

Fig. 1. Deployment of Mercury BLASTN on a diverse computer. Dashed lines and circles represent the locations of TimeTrial instrumentation taps.

measured. All four of these measurements require crossing timezones into a common virtual time.

The timezone properties of the experimental system are as follows. AMD Opteron processors running at 2.311 GHz are used as for all the cores. The processor cycle counts are converted to virtual time (i.e. nanoseconds) using a 0.43253 ns/cycle scale factor. FPGA cycle counts also need to be transformed from a 133 MHz clock rate. As a result, $s_P = 0.43253$ ns/cycle and $s_F = 7.5$ ns/cycle. The value of b_F was experimentally determined using the methodology presented above.

Figure 2 shows box-whisker plots of the above described latency measurements to compare each of 1000 48 kilo-base sequences against a 900 mega-base sequence. The latency profiles vary significantly between the four measurements. The first measurement has consistently low latency due to the virtual queue being empty when the first datum is sent. The second measurement has much higher latency because of the added delay from the filled virtual queues. The data-dependent flow throttling from the FPGA application adds to the spread in the latencies. The third measurement has the most variability of any of the measurements because the filtering nature of Mercury BLASTN which frequently does not produce hits for a large portion of the stream. The last datum returned from the FPGA has consistently low latency in large part because there is never significant data volume returning to the processor from the FPGA for this data set. As a result, this virtual queue is never full and the end of data marker that follows the last datum ensures that all the buffers in the path are immediately flushed.

IV. Conclusions

The TimeTrial performance monitor assists developers in understanding the executing dynamics of high performance streaming applications deployed on architecturally diverse platforms. To enable this understanding, it is often necessary to reason about time across platform boundaries. We have described an approach to time synchronization that is low overhead and enables time measurements between platforms. Future work will focus on the circumstance where the relative clock frequencies are not as well known a priori and extending these techniques to GPUs.

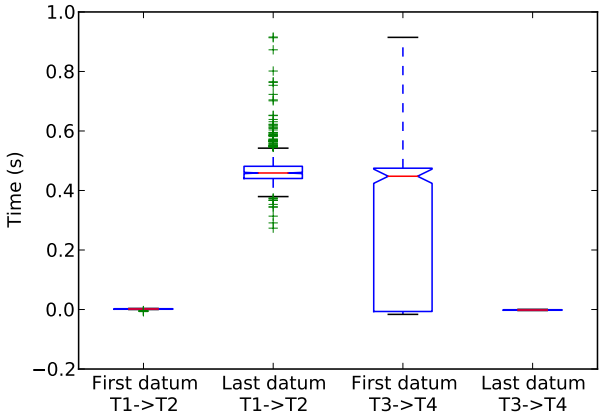

Fig. 2. Four latency measurements across virtual queues in Mercury BLASTN. The box-whisker plots indicate quartiles (within 1.5 interquartile range) and outliers.

Acknowledgments

This work was supported by the NSF under grants CNS-0931693, CNS-0905368, and CNS-0751212.

References

[1] J. Lancaster, J. Buhler, and R. D. Chamberlain, "Efficient runtime performance monitoring of FPGA-based applications," in *Proc. of 22nd IEEE Int'l System-on-Chip Conf.*, Sep. 2009, pp. 23–28.

[2] R. D. Chamberlain and J. M. Lancaster, "Better languages for more effective designing," in *Proc. of Int'l Conf. on Engineering of Reconfigurable Systems and Algorithms*, Jul. 2010.

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

[4] M. Genden *et al.*, "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.

[5] 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.

[6] J. D. Buhler, J. M. Lancaster, A. C. Jacob, and R. D. Chamberlain, "Mercury BLASTN: Faster DNA sequence comparison using a streaming hardware architecture," in *Proc. of Reconfigurable Systems Summer Institute*, Jul. 2007.

[7] P. Krishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang, A. Jacob, and J. Lancaster, "Biosequence similarity search on the *Mercury* system," *Journal of VLSI Signal Processing*, vol. 49, no. 1, pp. 101–121, Oct. 2007.

[8] J. Lancaster, J. Buhler, and R. D. Chamberlain, "Acceleration of ungapped extension in Mercury BLAST," *Journal of Microprocessors and Microsystems*, vol. 33, no. 4, pp. 281–289, Jun. 2009.