Runtime Performance Monitoring of Architecturally Diverse Systems

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

Introduction

Architecturally diverse computing systems (i.e., systems built from heterogeneous compute resources) are prevalent in HPEC environments due to the strict power and size limitations of the application domain. Utilizing the strengths of these different computing architectures can lead to greater power and size efficiencies relative to homogeneous systems alone. However, diverse computing systems present many application development challenges such as designing an application using many different architectures (e.g. x86/PowerPCs, DSPs, or FPGAs), partitioning the application across the different components, functionally debugging the application, and determining that the application meets performance requirements. In addition, these systems operate as a distributed system, where there is no global clock to order events.

This paper describes a novel runtime monitoring system that enables the developer to extract information from the execution of a streaming application. The runtime monitoring system evaluates performance expectations of a streaming application provided by the user to enable better system reliability and a shorter development cycle.

Streaming Programs

Generally, streaming applications can be thought of as course-grained dataflow computations in which computation blocks, or kernels, are interconnected by edges over which data is communicated. An example streaming application topology is illustrated in Figure 1. A data source delivers data to block A, the output data stream from block A is delivered as an input stream to block B and C, etc.

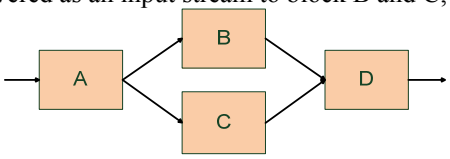

Figure 1: Example streaming application.

Dataflow is a natural computational model for reasoning about applications on diverse computing systems since the memory references are constrained to a *stream buffer* inside each kernel, the communication of information is explicit, and both wide and deep parallelism can be expressed in a straightforward manner. Since many diverse systems in the HPEC domain do not have shared memory, this encourages good memory partitioning of the application. This model also helps the developer reason about utilization of shared computational and interconnect resources since an explicit mapping of application kernels to compute resources and edges to interconnect resources must be performed to deploy the application on a real system.

Understanding Performance

For non-streaming applications on monolithic platforms, performance is typically understood via instrumentation that generates statistics at runtime (e.g. gprof). However, developing a reasonable global view of a distributed application's performance is not as straightforward. Distributed applications do not typically share storage, nor is there a universal program counter that accurately reflects the state of the application. A system-wide performance monitoring tool is essential to fully understand the characteristics of these applications.

Figure 2 shows an instrumentation of the sample streaming application described in Figure 1. The monitor observes communication of data on edges between kernels, treating the kernels as black boxes. This simplifies the implementation of the runtime monitor while providing much useful performance information. If more visibility is needed inside a kernel, the developer can simply provide an edge on the kernel boundary for the monitor.

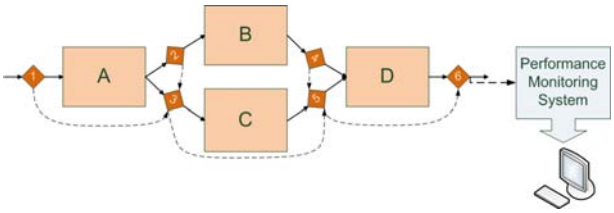

Figure 2: An example black-box monitor for the application in Figure 1. The edges between blocks are tappped to determine application bottlenecks.

Ideally, a performance monitoring system would provide timestamps for every event as data flows through the system, much like current simulations systems offer but operating at full system speed. Additionally, no overhead and perturbation of the system should occur so the measurements are perfectly accurate. Achieving this is impossible on most real systems as the bandwidth and storage are limited. Hence, any runtime system monitor must be carefully designed to minimize both the overhead and perturbation of the execution so as to not overwhelm the system with measurement data.

A popular approach to monitoring the performance of a program is to aggregate performance information over the entire run of a dataset. The aggregate information is then presented at the end of the run. This approach, however, misses a key piece of information: when do performance anomalies occur during the run? Revealing this information to real-time system developers can help them track down important performance anomalies that may not be present in aggregate statistics.

To provide temporal information, our monitor uses a technique based on the concept of frames from the media compression literature. Instead of measuring over the entire execution of a program, a measurement is divided into "frames." A frame for a kernel is defined as a segment of the execution given in time. Statistics are initialized each

time a new frame is encountered and the results are reported at the end of a frame. In this way, the system provides an ordered *set* of frames, each containing its measured performance, from which a developer can reason about the behavior of the application over time. Figure 3 shows an example visualization of a queue length, L_q, per frame.

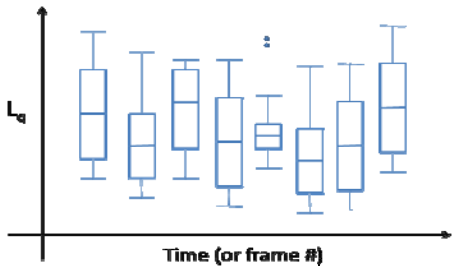

Figure 3: Box-plot summarizing the queue length for each frame. The middle bar represents the median and the horizontal bars denote the quartiles. Outliers are shown as dots.

Frames allow the developer to control the time resolution in which measurements are reported. Larger frames will reduce the bandwidth required for the monitoring task at the expense of temporal precision. While this is an essential first step in reducing the required bandwidth, the runtime monitor uses a number of additional data compression techniques. One of the most powerful compression tools of this monitoring system is the online evaluation of developer-driven data collection.

Developer-guided Performance Monitoring

A number of high-performing compression algorithms have been developed for reducing storage requirements of trace data. However, many of these have buffer requirements or computational complexity issues which make them unsuitable for HPEC environments. Trace compression also focuses heavily on lossless compression to support architectural simulation which may not be necessary for verifying performance. Instead of trying to create a compression algorithm that is targeted toward a small set of metrics (e.g., PC or instruction logging), this work utilizes input from the developer to guide compression. This enables performance measurements of interest to be collected with high fidelity while other measurements can be compressed more aggressively.

The monitoring system allows three categories of developer-guidance. First, the developer can specify a set of direct measurements to be extracted from the execution of the program. An example of this includes measuring the throughput of the edges in an application, or throughput only under certain conditions (e.g. when an input queue is not empty). Direct measurements are useful for developing a high-level view of the application performance.

Second, the developer can assert a number of performance expectations one has about the application. These expectations are described as a set of assertions which in turn are translated into predicates that the runtime system can evaluate. If the predicates hold, the developer then knows that the system is meeting expectations, at least for the dataset that was executed. When one or more assertions fail, there are a number of ways to proceed. A sensible way for soft constraints is to simply report the number of frames which it held. For more serious performance issues, the failure mode can be to dump trace data to give simulator-like visibility to help diagnose the anomaly.

The third category of developer-guidance is performance expectations that are conditioned by statistical measures. For instance, one might want to know whether or not the occupancy of a queue matches a certain distribution. Since this is an empirical measurement, a Q-Q test can be used to determine how closely the queue occupancy matches the theoretical distribution. To evaluate whether this type of predicate holds, a statistical threshold of similarity needs to be specified.

In order to realize such a system, a simple and expressive language needs to be utilized to articulate the performance expectations. Several languages have been proposed for evaluating functional correctness of real time systems, such as Linear Temporal Logic and Interval Temporal Logic. The first implementation of the performance monitor utilizes Logic of Constraints (LoC) [1] as the assertion input mechanism. LoC is a formal language which supports describing both performance and functional predicates. An example performance assertion of the throughput of block A in Figure 1 can be stated informally as: "at least 10 *A.OutputData* events will be produced in any period of 1000 time units." The corresponding LoC predicate can be written as:

$$t(A.OutputData[i+10]) - t(A.OutputData[i]) \leq 1000$$

Some related research into user-guided performance monitoring exists [2, 3, 4]. In [2], logs are produced which can be analyzed offline and their approach is not appropriate for concurrent systems. In [3], performance assertions can be evaluated for constraints which touch only a single process, which is very limiting for HPEC environments. [4] is targeted toward mobile devices and they extend the work in [3] to evaluate assertions across multiple processes. However, none of these proposals work in an architecturally-diverse computing platform, are not targeted toward a streaming programming model, and do not provide the ability to evaluate system-level performance assertions.

Conclusions

The need to better understand the performance in HPEC systems is clear. The runtime performance monitor described in this paper will help this task by providing both measurements and performance verification of an application executing a real-world dataset.

[1] Xi Chen, Hsieh, H., Balarin, F. and Watanabe, Y. "Logic of constraints: a quantitative performance and functional constraint formalism," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol.23, no.8, pp. 1243-1255, Aug. 2004.

[2] Perl, S. E. and Weihl, W. E. "Performance assertion checking," *SIGOPS Oper. Syst. Rev.* 27, no. 5, pp. 134-145, Dec. 1993.

[3] Vetter, J. S. and Worley, P. H. "Asserting performance expectations," In *Proc. of ACM/IEEE Conference on Supercomputing*, pp. 1-13, 2002.

[4] Lencevicius, R. and Metz, E. "Performance assertions for mobile devices," In *Proc. of Int'l Symposium on Software Testing and Analysis*, pp. 225-232, July 2006.