Schedulers as Model-Based Design Elements in Programmable Heterogeneous Multiprocessors

JoAnn M. Paul, Alex Bobrek, Jeffrey E. Nelson, Joshua J. Pieper, Donald E. Thomas
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{jpaul, abobrek, jnelson, jpieper, thomas} @ ece.cmu.edu

Abstract

As System On a Chip (SoC) designs become more like Programmable Heterogeneous Multiprocessors (PHMs), the highest levels of design will place emphasis on the custom design of elements that were traditionally associated with systems in the large. We motivate how schedulers that make dynamic, data-dependent decisions at run-time will be key design elements in PHM SoCs. Starting from a fundamental model, the role schedulers play in PHMs is developed. Model-based scheduling is introduced as an approach to designing schedulers that optimize a PHM's performance. Due to the complexity of the PHM design space, convergence on optimal design requires high-level modeling and simulation. In model-based scheduling, high-level models of scheduling decisions result in actual design elements that appear in real systems. Experiments for a simple two-processor PHM that does a mix of image and text compression are included. Results show the effectiveness of model-based scheduling.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: *Modeling techniques, Performance attributes.* I.6.5 [**Simulation and Modeling**]: Model Development — *Modeling Methodologies.*

General Terms. Performance, Design.

Keywords. Computer-Aided Design, Performance Modeling, System Modeling, Schedulers, Heterogeneous Multiprocessors.

1. Introduction

Enabling the potential of billion-transistor systems-on-chip (SoC) designs will necessitate a quantum leap in design methodology. As computation leaves the desktop and ubiquitous computing becomes a reality, performance-based design of concurrent software executing on concurrent hardware will be the future. Resultant devices will be space-constrained and will simultaneously interact with humans, the physical world, and other computers. They will include increasingly sophisticated software functionality that cannot be designed using approaches that were intended to meet only the challenges of purely reactive, embedded systems. The central challenge in future computer design is defining new design elements, programmer's views, simulation foundations, and languages that permit the designer to model and manipulate how the interaction and co-operation of multiple programmable elements affect system performance.

Many agree that future SoC designs will be heterogeneous multiprocessors [1], while network-on-a-chip communication suggests relatively coarse-grained design elements communicating asynchronously in a data-dependent manner. Instruction Set Simulator (ISS) models do not allow exploration of a such a broad and complex design space. Rather, high-level modeling and simulation must define new design elements that capture the interactions of software executing *on* and *across* concurrent programmable resources, capture how software loads resources, capture performance of data-dependent execution, and not over-restrict the design space. Further, new forms of design trade-offs must be invented and supported for these new design elements, for example, trading-off scheduling decisions for numbers and types of processing resources.

We introduce the concept of SoCs as *Programmable Heterogeneous Multiprocessors* (PHMs), emphasizing that not only will individual processors be programmable, but the system formed by the interaction of the processors must also be considered programmable. We present the role of custom schedulers as design entities. For instance, such schedulers decide in a data-dependent manner which thread executes next or which packet is transferred next. We consider the design of a scheduler's functionality to be based upon a reduced-detail model of the characteristics of the remainder of the PHM with which the scheduler interacts. We refer to this approach as *model-based scheduling.* Because the schedulers are developed to models using simulation at design time, yet still make dynamic, data-dependent decisions at run-time, model-based scheduling is a mix of static and dynamic decision-making. We motivate and illustrate through an example how aspects of system modeling determined at design time can lead to more effective run time scheduling decisions.

2. Scheduler Relationships

Schedulers co-ordinate the interaction and execution of multiple elements in a computer system, thus their design is central to maximizing the performance of a PHM. In our Modeling Environment for Software and Hardware (MESH), schedulers are design elements. Unlike the role of commercial operating systems or real-time schedulers, our schedulers can be simple decision-making entities, selecting which thread will execute next or which packet will be sent next. Because these decisions are customized to the way the anticipated applications will execute on characteristics of the underlying hardware, our schedulers are modeling elements that are literally on the boundary between high-level models and actual system elements. They provide the means to capture concurrent model interactions, and serve to resolve the computational work being consumed by the software in the system.

2.1 Reactive and Interactive Relationships

We begin by developing properties of schedulers as layered relationships. In a synchronous FSM design, the external environment is modeled as a testbench (TB) that conceptually shares the same clock with the FSM. The TB exercises inputs and reads outputs of the FSM in synchrony with the FSM. The scheduling of the system is constrained as a *reaction* within a fixed number of time ticks to timed data values presented by the TB. The FSM may be implemented either in hardware or in processor systems with static, data-independent schedules and response times. What is important is the fixed response time of the FSM as it reacts to data that is presented to it by its environment.

In Figure 1, a request/acknowledge signal between an FSM and TB enables a data-dependent, self-timed form of computing [2]. The classic handshaking signals enable the FSM to have computation times which vary with the values of the data it receives. Thus the FSM no longer reacts to the TB, but now *interacts* with it. The clock reference no longer controls the timing of the system as a whole; computation completion time is now data-dependent.

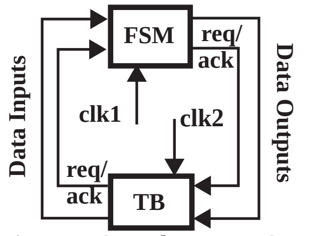

Figure 1 Asynchronous FSM-TB Interactions

There are three key observations that can be made from this example. The first is that the simple handshaking protocol between the two machines is a layer of scheduling that exists *above* them, providing services *to* them. The scheduling layer does not contain the FSM and TB in a component-like fashion as if it were some higher-level description of them. Secondly, the overall performance of the system is dependent on the scheduling decision between the FSM and TB (however simple in this case), the actual data values, and the execution times resulting from the separate clock speeds.

Third, because more complex interactions between the FSM and the TB are allowed, the FSM can take on processor-like properties. As a programmed processing element (PE) the FSM block can now defer interaction with the TB should it choose to perform, for instance, additional local processing between requests to the TB. The FSM/PE may even be a multithreaded system where local tasks compete for the FSM/PE resource with tasks that process the interaction with the TB. In such a case there are overlapping schedulers, the req/ack and the multithreaded FSM/PE scheduler. Thus, complex local scheduling decisions in the FSM/PE also contribute to the overall system performance.

2.2 Heterogeneous Multiprocessing

A very general version of a PHM is shown in Figure 2. The FSMs of Figure 1 are now labeled as processing elements (PE), acknowledging their programmable nature. System I/O is co-coordinated to the TB, which also maintains the timing reference for the system as clktb. Each PE in the system may have a separate clock. Processing elements interact by a conceptually global, coordinating control mechanism, labeled, "scheduling and arbitration" (SA). In Figure 1, point-to-point interfaces resolve the scheduling between two elements. By contrast the SA of Figure 2 permits arbitrary numbers of elements to be co-coordinated using scheduling elements distributed across the PEs. As in Figure 1, the SA is conceptually layered onto the PEs of the system. It co-ordinates, but does not contain the system resources. Due to this relationship and unlike the traditional hardware design elements,

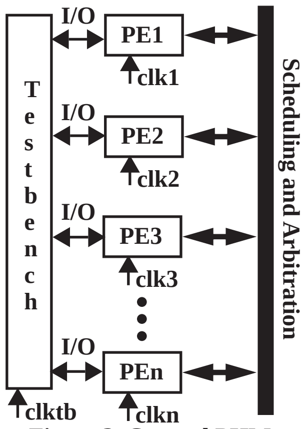

Figure 2 General PHM

design elements of a PHM are heterogeneous and layered. They interact in complex ways as overlapping scheduling and programming domains compete for shared system resources. For effective performance design of the system of Figure 2, model complexity must be reduced in a meaningful way that does not miss these interactions.

3. Schedulers as Design Elements

Schedulers are important elements of a programmable, concurrent design; they must be modeled and developed in conjunction with other design elements for effective design of PHMs. They, in effect, manifest the global system control flow across system resources, i.e. distributed control flow. Schedulers are layered abstractions. The layering provides the basis for the global extent of the performance impacts of local scheduling decisions; schedulers reside on individual processing resources, even if they logically co-operate in a global context. Models of high-level, machine-specific, data-dependent scheduling decisions lead to performance optimization.

3.1 Intra- and Inter-Resource Scheduling

In Figure 3, the layered multithreading view of MESH is shown for a single resource in a multi-resource system such as a single PE of Figure 2. At the top of the slice is a dynamic number of unrestricted software threads (labeled as logical threads $Th_{Li1} \ldots Th_{Lin}$). These are interleaved for execution on the resource labeled as a physical thread, Th_{Pi}, by a scheduler U_{Pi} which resides on Th_{Pi}. U_{Pi} threads determine intra-resource timing of software executing on hardware by selecting which thread to execute and determining how long it is permitted to execute before the next resource is interleaved in the simulation. Thus, software execution is modeled in both a data and resource dependent manner. U_{Li} threads can also logically group resources for inter-resource scheduling as shown by the dashed oval going off of the figure. This permits M threads to be dynamically mapped to N resources, e.g., a pthread scheduler. This layer between the resource and software models brings together physically timed resource models and self-timed, (logically) interacting dynamic software threads [3].

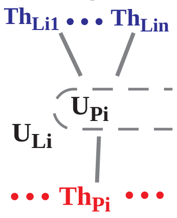

Figure 3 A Slice of the Layering

In MESH, the intra-resource physical timing of software executing on hardware is done by using designer-instrumented "consume call" annotations that indicate computational complexity of software regions. The resource thread (Th_{Pi}) provides a computing budget to the U_{Pi} scheduler at a specified rate. The U_{Pi} scheduler allows its threads to execute until that resource budget is consumed. At this point, other resource threads may execute similarly. If the resource budget or the rate of its execution (based on physical time) is increased, the resource power is increased; more state is advanced by that resource relative to other resources in the system. Thus, the state advancement per unit time (performance) is dependent on resource budgets and rates, software complexity as modeled by consume calls, scheduling state, and inter-resource, concurrent interactions.

Layering is what distinguishes MESH from other approaches. Software is inherently different from hardware [4]; intuitively software is layered onto hardware. Component views capture static forms of scheduling [5]. Other approaches that seek to capture performance modeling for hardware/software co-design are focused on the resolution of different timing models [6], but do not capture software, schedulers and hardware as layered design elements that lead to performance effects. We have done prior work in MESH, showing how the design space of a network processor with software, hardware and scheduling decisions allow for more abstract, high level modeling to capture the design space that might otherwise only be captured in a detailed ISS [3].

The model of the layered, intra-resource scheduling of Figure 3 is represented in the PHM model of Figure 4. The scheduler on the processor modeled as physical thread Th_{P1}, is shown highlighted in the gray box as scheduler U_{P1}. The schedulers, U_{P1}, U_{P2},… U_{Pn}, make decisions about which software threads are executed on the individual resources. Importantly, they can do so either naïvely or by more machine-specific forms of decision making.

Dynamic decisions made when one or more threads are eligible to execute on a given processor must be made with respect to some context. When the context is purely local and machine-unspecific, naïve scheduling decisions such as round-robin strategies result. Scheduling decisions made in the context of additional system information, both static and dynamic, can have great impact on overall system performance [7]. Inter-resource scheduling creates a design layer above that of the software executing on an individual processor.

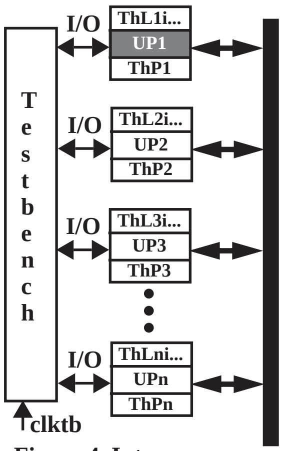

Figure 4 Inter-resource Scheduling in a PHM

For large PHM systems designers will have design-time system-specific information which will allow for more effective local, dynamic scheduling decisions to be made in the context of the way a resource interacts with the rest of the system. This forms the basis of effective inter-resource scheduling in a PHM and thus the basis of distributed global control in a PHM.

3.2 Model-Based Schedulers

When simulation-based techniques are utilized at design time, inter-resource, data-dependent optimizations can be derived. Local resources can utilize high-level functional models of, and possibly state from, all or a portion of the remainder of the PHM to make data-dependent dynamic scheduling decisions. These models lead to scheduler functionality that resides on the actual resources in the system. The actual scheduling decisions are made dynamically, i.e. at run time; the scheduler functionality is derived from design time models. We define this approach as *model-based scheduling.*

The functional models of the scheduling decisions are designed in the context of machine-specific high-level models of the remainder of the PHM system, its data-dependent scenarios, and the way the system will be exercised. A key question faced by designers of model-based schedulers is how much system information is required to make effective scheduling decisions. Since the complexity of computing a scheduling decision and the amount of information needed clearly impacts the time it takes to make a scheduling decision, designers must strike a balance between too little and too much information. MESH facilitates the development of model-based schedulers that strike this balance.

4. Illustration

Our illustration will show how model-based scheduling decisions can impact the overall performance of a PHM. We consider a simple multimedia system consisting of two heterogeneous processors (a Mitsubishi M32R and an ARM) to handle a mixture of image and text compression applications. The image compression algorithm includes a wavelet transform and zerotree quantization. The wavelet step transforms the image into a series of frequency sub-bands. Zerotree quantization then selects data to discard using control-intensive algorithms. Text compression is performed using the gzip program [8] that looks for redundant data patterns in plaintext, mainly stressing memory bandwidth and latency.

Due to their unique characteristics, each processor in this system can execute the three tasks at different rates relative to one another. For example, the M32R contains a multiply-accumulate (MAC) instruction commonly found in DSP processors that make it better suited for performing wavelet transforms. While both processors are equally adept at quantization, the ARM is approximately twice as fast at gzip text compression, and the M32R is approximately twice as fast at the wavelet transform.

4.1 Scheduling Trade-offs

In order to maximize the performance of this heterogeneous system, we examine several schedulers: static (U_S), system load-aware (U_{LA}), and system architecture-aware (U_{AA}). Figure 5 shows relationships between the scheduling strategies; all use the dotted lines, other lines differentiate strategies. The static scheduler performs the wavelet transform tasks only on the M32R while the quantization and text compression tasks are executed on the ARM (solid black lines). This naïve strategy may perform well for known mixtures of image and text data inputs, but corner cases exist where the system performance suffers.

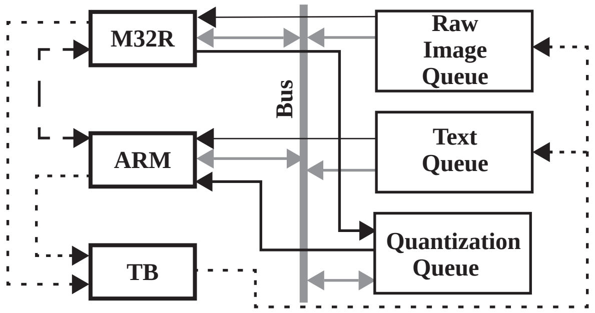

Figure 5 Flow Relationships For All Schedulers

To keep the system resources utilized, each processor's scheduler must be aware of the system load and be able to schedule any task type on any resource. The load-aware schedulers (U_{LA}, gray bus and arrows) of each resource monitor the status of all three task queues, selecting the next job based on the queue occupation. When a queue size is backed up, the schedulers will map jobs onto inappropriate resources (wavelet transform on ARM or text compression on M32R) to maintain high performance.

Even though the U_{LA} schedulers can make decisions based on pending tasks, they are not aware of how adept a given processor is at handling each task. The U_{LA} scheduler is not designed around the individual processor's capabilities; it is not a model-based scheduler. However, the architecture-aware scheduler (U_{AA}, dashed line and gray bus and arrows) has performance information for the M32R and ARM. The U_{AA} scheduler can be thought of as a superset of the U_{LA} scheduler where each local scheduler on the M32R and ARM maintains a model of the remaining processing time of the other resources (dashed line between processors). This model is used to determine if mapping a packet onto an idle inappropriate resource, rather than waiting for the faster processor, would result in a lower overall latency. When the queue sizes reach a certain threshold, the U_{AA} scheduler defaults to the U_{LA} behavior of taking all packets. While the U_{AA} scheduler is a model-based scheduler, the U_{AA} and U_{LA} schedulers also overlap.

To provide a basis for comparison, we constructed a cycle accurate instruction set simulator for the combined system. The ISS consists of the GNU ARM and M32R simulators [8], connected via shared memory. The two processors run in lockstep with time advancing one cycle at a time. Scheduling strategies used in the ISS are identical to those used in the MESH.

4.2 High-level Thread Relationships

The image/text compression system was also modeled using the thread relationships in MESH as shown in Figure 6.

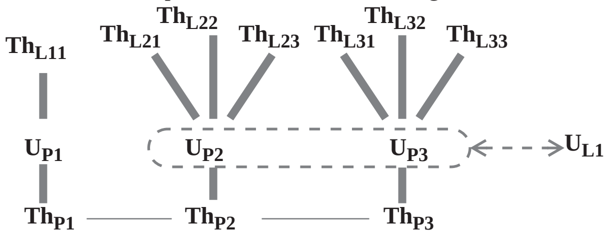

Figure 6 High-Level Thread Relationships in MESH

Physical threads Th_{P1}, Th_{P2} and Th_{P3} model the high-level, rate-based interleaving of the processor resources. Th_{P1} models the testbench resource, Th_{P2} models the M32R and Th_{P3} models the ARM. In addition to resolving computation consumption to resource power, U_{P2} and U_{P3} also make decisions about which of three possible threads execute on the resource they reside on, i.e., the Th_{Lij} on processor i. The threads represent the wavelet transform, quantization, and text compression functionality for each processor. U_{P2} and U_{P3} conceptually form the global, logical scheduling context, U_{L1}, which arises from the co-operation of local scheduling decisions on individual processor resources.

We developed three separate sets of U_{P2} and U_{P3} schedulers, showing the effectiveness of model-based scheduling: static (U_S), system load-aware (U_{LA}), and system architecture-aware (U_{AA}).

4.3 Experiments

The testbench inputs compression packets at a Poisson rate to the system in order to explore a wide variety of data dependent effects. Performance was measured while changing the image to text packet ratio as well as varying the packet size. Results are the greatest system throughput at a fixed maximum latency.

Figure 7 compares the performance of the U_S and U_{LA} schedulers in ISS and MESH simulation environments while the ratio of text to image jobs is varied. The U_{LA} scheduler improves performance of jobs with high amount of text content because it allows for the M32R to take on text compression jobs in the absence of images. The MESH results match well with their ISS counterparts showing that MESH is able to accurately simulate the system while abstracting much of the hardware detail.

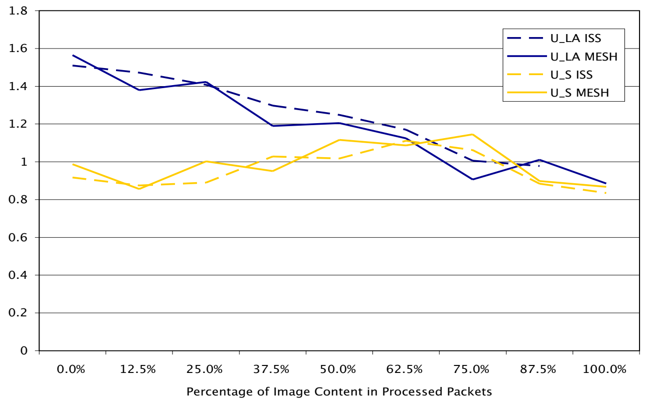

Figure 7 U_S and U_{LA} Schedulers, ISS and MESH Models

We explored the design space further by considering a scenario of image and text input files of widely varying sizes with a fixed ratio. A third, model-based scheduler, U_{AA}, intelligently schedules jobs on inappropriate resources sooner. The U_{AA} scheduler showed a minimum improvement of 9%, average improvement of 19%, and maximum improvement of 40% in the number of kilobytes processed per million ISS cycles as compared to the U_{LA} scheduler. Even in this simple example, the interactions between resources and schedulers were very complex; analytical solutions were not feasible. ISS-level simulations with full software models are impractical for exploration of future, more complex designs. Significantly, while MESH supports reduced detail, high-level models it also provides the basis for new design elements such as model-based schedulers that appear in real systems.

5. Conclusion

By developing the role of schedulers in PHMs, we motivate how they may be optimized for dynamic, data-dependent execution over a set of resources and an intended application space. Model-based schedulers were introduced as elements in high-level models of PHMs. Significantly, they can also result in actual design elements that appear in systems. They allow for customization of the dynamic, data-dependent local scheduling decisions that individual processors make. The customization can be thought of as high-level models of physical properties of the machine with which the processor interacts. MESH was shown to provide efficient exploration of the effectiveness of scheduling decisions on overall system performance. Our experiments for a simple two-processor PHM that does a mix of image and text compression illustrate the effectiveness of considering model-based schedulers as design elements that must be explored at the high-level along with high-level models of concurrent software executing on concurrent hardware in PHMs.

6. Acknowledgements

This work was supported in part by ST Microelectronics, General Motors, an NSF Graduate Research Fellowship, and the National Science Foundation under Grant 0103706. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

7. References

[1] "Are Single-Chip Multiprocessors in Reach?", *IEEE Design & Test*, Jan - Feb 2001.

[2] C.L. Seitz. "System Timing." *Introduction to VLSI Systems*. C. Mead, L. Conway. Reading, MA: Addison-Wesley, 1980.

[3] A.S. Cassidy, J.M.Paul, D.E.Thomas."Layered, Multi-Threaded, High-Level Performance Design. *DATE* 2003

[4] B. Grattan, G. Stitt, F. Vahid. "Codesign-extended applications." *CODES* 2002.

[5] F. Gharsalli, S. Meftali, F. Rousseau, A. Jerraya. "Automatic generation of embedded memory wrapper for multiprocessor SoC," *DAC* 2002.

[6] K. Richter, R. Ernst. "Event model interfaces for heterogeneous system analysis," *DATE* 2002.

[7] D. Skillcorn, D. Talia. Models and Languages for Parallel Computation, *ACM Computing Surveys*. 1998.

[8] http://www.gnu.org