

Benchmark-Based Design Strategies for Single Chip Heterogeneous Multiprocessors

JoAnn M. Paul, Donald E. Thomas, Alex Bobrek

Electrical and Computer Engineering Department

Carnegie Mellon University

Pittsburgh, PA 15213 USA

{jpaul, thomas, abobrek}@ece.cmu.edu

Abstract

Single chip heterogeneous multiprocessors are arising to meet the computational demands of portable and handheld devices. These computing systems are not fully custom designs traditionally targeted by the Design Automation (DA) community, general purpose designs traditionally targeted by the Computer Architecture (CA) community, nor pure embedded designs traditionally targeted by the real-time (RT) community. An entirely new design philosophy will be needed for this hybrid class of computing. The programming of the device will be drawn from a narrower set of applications with execution that persists in the system over a longer period of time than for general purpose programming. But the devices will still be programmable, not only at the level of the individual processing element, but across multiple processing elements and even the entire chip. The design of other programmable single chip computers has enjoyed an era where the design trade-offs could be captured in simulators such as SimpleScalar and performance could be evaluated to the SPEC benchmarks. Motivated by this, we describe new benchmark-based design strategies for single chip heterogeneous multiprocessors. We include an example and results.

Categories and Subject Descriptors

C.4 [Computer Systems Organization]: Performance of Systems – *Modeling Techniques*.

General Terms

Performance, Design.

Keywords

Heterogeneous Multiprocessing, Systems-on-Chips (SoCs), Benchmarking, Scenario-oriented Design.

1. The Next Generation of Computing

Computing will leave the traditional categories of desktop, embedded and portable to become more ubiquitous and pervasive [1]. In order to meet performance, power and space requirements while retaining the flexibility of programmable devices, single chip computers will take on the appearance of heterogeneous multiprocessors [2], creating a new breed of design. The possibilities for this next generation of computing are limited by the ability to design them with an economically practical amount of design effort [3]. The next generation of computing is not merely the next evolutionary step in computing. New design strategies are required [4] beyond that of even symmetric multiprocessing, which is focused on single, concurrent applications [5]. The applications

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CODES+ISSS'04, September 8-10, 2004, Stockholm, Sweden.
Copyright 2004 ACM 1-58113-937-3/04/0009...\$5.00.

executing on these single chip computing devices will be comprised of heterogeneous, concurrent application elements with a mix of timing and performance requirements.

Individual, programmable processors that carry out heterogeneous, concurrent applications will be like registers were to early VLSI design — building blocks within a larger organizing framework. And yet those same processor building blocks will be differentiated from each other in the capabilities of their hardware, the way they are programmed, and their manner of interconnect. We refer to this next generation of design as that of Programmable Heterogeneous Multiprocessors (PHMs), emphasizing that not only will individual processing elements on the device be programmable, but the entire chip must be considered programmable as well.

Key to the development of new programming strategies is the ability to evaluate programmable designs. We introduce *scenarios* as a new form of benchmark-based design evaluation for next generation designs that captures the need to mix timed and untimed I/Os, and evaluate concurrent applications executing on heterogeneous architectures. Scenarios are developed by comparing the contrasting design philosophies of Computer-Aided Design and Computer Architecture. Through an example, we describe how scenarios enable the performance evaluation of PHMs in a way traditional approaches cannot.

2. The Landscape of Design

Future single chip computer systems will be hybrids of two very different classes of design, that of general purpose, programmable systems and application-specific, embedded systems.

In the first, individual programs are the primary system input — the program comes from “outside the box” being designed and is executed by it. As a result, performance of the architecture is evaluated to an anticipated set of programs — or benchmarks — most typically executed individually. In the second, a data stream is the primary input. The system’s application, often repeatedly executed in a loop, is designed to be “inside the box” with the constraint that it meets the real-time demands of the input stream.

In this section we motivate a new class of computing as a hybrid of design by considering the contrast between the design automation and computer architecture research communities. While the objective of these two communities is the design of computing systems, they represent very different philosophies.

2.1 The Design Automation Approach

The basic objective of the Design Automation (DA) community is the design of an instance of a computer system given an up-front specification of the application it will carry out (Figure 1, left). These systems are designed to meet real-time demands of processing an I/O stream, placing emphasis not only on the data received, but on the timing of the data as well. Therefore, the DA approach I/O stream can be well represented as a set of {value, time} tuples. This basic philosophy is drawn from a model of a finite state machine and captured in design languages such as

Verilog and VHDL. These simulation-based models interact with testbenches that model the external system with which the system under design is to continuously interact. The specified applications are executed repeatedly for long periods of time, processing many different input sets to produce many output sets.

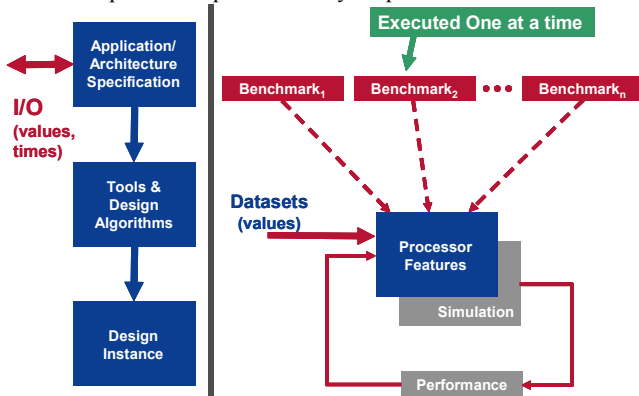


Figure 1 Design Automation (DA), on Left, and Computer Architecture (CA), on Right

Emphasis is placed on meeting the demands of the I/O stream by direct manipulation of the parallelism within a design, ranging from the number of bits in a word to the number of processors in the system. However, emphasis is not placed on optimizing the execution bandwidth of programs or applications unknown at design time. Conceptually, these are not inputs to these systems. If portions of the system are programmed, the programs are part of the specification, literally functionality that is to be included “inside the box.”

Designers specify the application up front in some language so that as many detailed design decisions as possible can be taken out of the designer’s hands. This is a top-down style of design which emphasizes specification for automatic synthesis as shown in Figure 1 on the left. This view captures the potential for refinement through iteration, and the fine-tuning of the design instance to difficult-to-meet and often competing physical constraints, e.g., performance, power and area.

Because of the up front specification of the design application, the DA community has focused on the challenges of designing fully custom hardware devices (Application Specific Integrated Circuits — ASICS), and more recently, embedded systems that may include one or more programmed processing elements. A traditional application area for ASICS is in the set of consumer-oriented applications for multimedia, where compute intensive signal and image processing applications need to be executed fast and in small devices that consume small amounts of power.

“Embedded systems” is a phrase which used to apply to the design of any computer embedded in a non-computer system. The need for fixed and guaranteed response times created the real-time (RT) class of design emphasizing the scheduling of a set of programs onto a shared processing element. The need to meet worst-case, real-time computing demands again presumes that the application is given up-front, where the emphasis is placed on analytical modeling in a top-down design style.

More recently, the design automation of embedded systems has taken on the terminology of “hardware/software codesign.” Some coin the design of these systems as “function/architecture codesign,” [6] implying that an initial specification includes two separate specification style inputs, the application as well as the architecture. But the basic design philosophy remains the same.

2.2 The Computer Architecture Approach

In contrast, the Computer Architecture (CA) design community

uses a simulation foundation for the optimization of architectural features of programmable systems. Programs are no longer “inside the box” and considered a part of the design, but are inputs to the design (Figure 1, right). Indeed, a processor will be used to execute numerous programs over its lifetime, many of which weren’t even conceived when the processor was designed. The execution bandwidth of these programs is of primary concern, creating a distinctly different design philosophy from that of DA.

The design of the creative features of a processor, such as caches and branch predictors, are evaluated using simulators that execute a standard set of programs which come from benchmark suites. A design flow might include the SimpleScalar [7] simulator and SPEC [8] benchmark suite. Performance is then evaluated from the individual software programs of the benchmark suite as they are executed one at a time on the computer architecture. Datasets are also a part of the SPEC benchmark suite, but the data is limited to a set of values designed to capture different executions of programs as they are executed one at a time. The inputs are not presented to a system in a timed fashion. Thus inputs in the CA design style are shown as untimed values which are presumed to be retrieved by a program as it executes. Using SPEC, designers evaluate if a given processor or processor feature improves performance for one, some, or all of the programs in the benchmark suite. Compilers can also be designed and evaluated using benchmark suites as they execute on processor models.

2.3 A New Approach

Our goal is the design methodology for a new, hybrid class of single chip design, that of Programmable Heterogeneous Multiprocessors where the chip is viewed as a *programmable collection of processors*. In PHMs, groups of processors can be programmed (i.e., scheduled) to execute sets of concurrent applications that respond to timed input datasets.

From the CA community we use the concept that a common simulation environment for architectures is the best way to focus on design trade-offs that support classes of programs. The design of these systems will require performance and power modeling above the instruction set simulation (ISS) level of the heterogeneous processors, software tasks, the scheduling of software tasks, and the interconnection network in a manner that will enable the exploration of a system’s design space. We used such a simulation environment, discussed in Section 4.1.

Also from the CA community, we use the concept that benchmark-programs will be an important class of inputs used to evaluate the programmable aspects of the design. As compared to SPEC, embedded system benchmarks are of more interest to PHM design. Several benchmark suites have been developed for this design space, including EEMBC [9], BDTI [10], MiBench [11] and MediaBench [12]. Yet these benchmarks are designed for single processor systems; they evaluate a single program at a time, not programs concurrently executing in multiple collections under a variety of situations across a variety of resources.

From the DA community, we add to the system under design a timed I/O stream as another primary input used to evaluate the design. Thus our designs include the timing requirements of the DA approach as well as the general programmability of the CA approach. Accordingly, a hybrid design style is born.

Figure 2 illustrates our approach. At the heart of the figure is a high-level model of a heterogeneous multiprocessor, designed to a simulation foundation. Nine processing elements (PEs) are illustrated with the implication that tens or hundreds of heterogeneous PEs, memories and interconnects may be manipulated in these designs. The benchmarks are now grouped into concurrent, heterogeneous collections recognizing that we are designing a single chip with concurrent hardware and software.

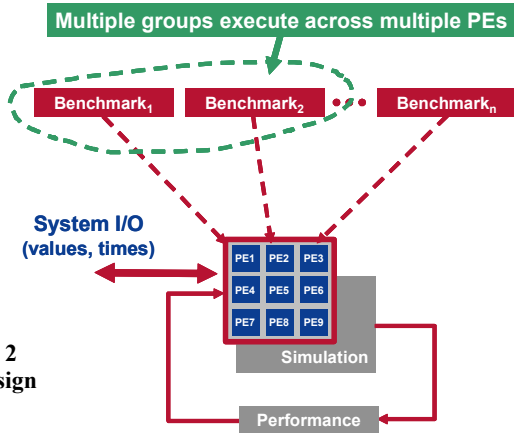


Figure 2 PHM Design

Overall performance evaluation of the chip will, in general, be more complex than for pure DA or pure CA designs. For example, CA might use a single benchmark runtime to evaluate the performance of a processor, while DA might use the number of gates in a design as a metric. Instead, a PHM must be evaluated against anticipated sets of heterogeneous applications executing in groups over datasets that exercise the system over long periods of time. Performance is now evaluated over a set of conventional programs that concurrently execute.

3. New Benchmarks — Scenarios

Benchmarks serve as a standard reference against which many designs can be evaluated. *Scenarios* extend the notion of a benchmark for single chip heterogeneous multiprocessors. Scenarios are comprised of three main parts: application groupings, system I/O, and a scenario program. The application groupings are formed from application elements that utilize conventional programmability of the CA approach, the I/O datasets introduce system-level timing demands from the DA approach, and the scenario programs schedule the application groupings. Figure 3 shows how the parts of a scenario relate to a PHM simulator of a heterogeneous multiprocessor architecture.

The figure also shows *coordinating programs* which we define as a scheduler class that uses the state of the *actual system* to optimize the execution of application elements on the chip resources. Unlike scenario programs, discussed in Section 3.3, coordinating programs operate at a physical layer of scheduling and are aware of architectural details. Unlike scenario programs, they are a part of the PHM undergoing evaluation by the scenario.

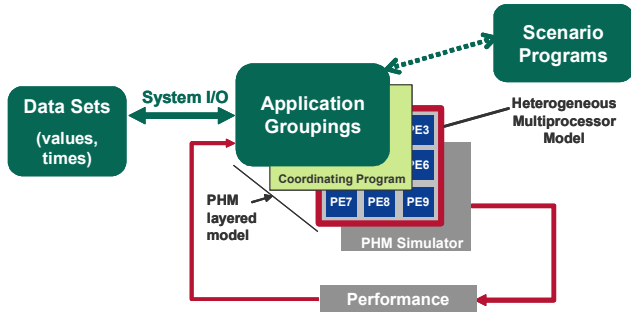


Figure 3 Scenarios Used to Evaluate PHMs

3.1 Application Groupings

Application groupings are a set of concurrent applications. In general, these have a high degree of independence but are considered together to serve external demand. Applications are formed from application elements as illustrated in Figure 4; the

logical dependencies between application elements serve to form an application. Application elements are analogous to the programs of a traditional benchmark suite, but more specific to the types of applications that will execute on portable and handheld devices. The reason for making the smallest unit of granularity smaller than an application is twofold. First, the application elements are likely to be re-used in many common applications. Second, many applications are composed of smaller units of functionality which are inherently parallelizable, either spatially concurrent or pipelineable, enabling the potential for a rich set of scheduling and architectural trade-offs for the same scenario.

As an illustration, we will combine several existing application elements, which are programs from the MiBench benchmark suite, into application groupings that might be found in a cell phone (Figure 4). A wide variety of MiBench benchmarks are used, including JPEG image compression, cyclic redundancy check (CRC), MP3 decoding algorithm (mad), rijndael encryption, adaptive pulse code modulation (ADPCM) which is used to compress voice data, and speech synthesis (rsynth).

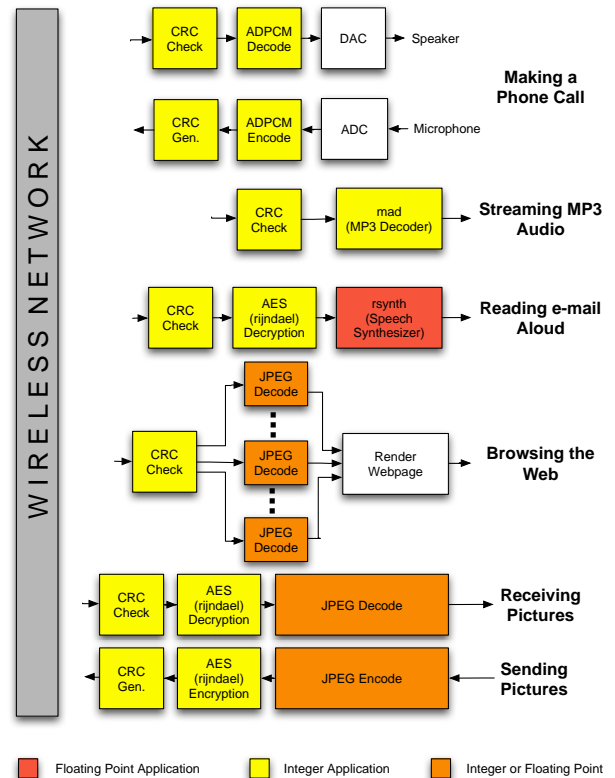


Figure 4 Example Applications of a Cell Phone

Figure 4 shows the arrangement of the benchmark elements into five applications. The left side of the figure represents the outside world (accessed via a wireless network), while the right side of the figure represents the data sent to the user (via display, speakers, etc.). The arrows between the benchmark elements show the direction of data flow for each application.

In Figure 4, some applications can be executed at the same time while other pairings do not make logical sense. For example, *streaming_mp3_audio* and *reading_e-mail_aloud* can never be used simultaneously because a user cannot listen to two audio streams at the same time. However, it could be possible to use the phone call functionality with reading e-mail where the user might want to share the contents of an e-mail over the phone. One might

be prioritized over the other, allowing resources to be shared for multiple, anticipated situations in a programmable design.

3.2 System I/O

The *system I/O* of Figure 3 models all of the external systems with which the PHM interacts. The {value, time} tuples create a processing demand on the system by the external environment. This demand may be continuous and periodic, typical of many embedded real-time systems, or aperiodic, representing sporadic system requests. It may also model the availability of data that the system under design (the programmed PHM) requests from an external system, such as a web server.

Next generation systems will not be designed to accommodate worst-case behavior of all possible application groupings. Rather, they must respond to a mix of behavior types, with performance evaluation taking place over a far longer period of time than traditional benchmarks. Further, the system I/O must exercise relevant corner cases of data and application groupings as applications compete for system resources. Thus, the size, values and arrival rates of I/O datasets are an important part of scenarios.

3.3 Scenario Programs

Even in this relatively straightforward cell phone example, it is easy to see how applications can be combined to satisfy the functionality of different situations. Consider an “airport scenario” where a business user is at an airport using the cell phone to read_e-mails_aloud, browse_the_web, and receive_pictures from a business associate. Next, consider a “bragging grandma” scenario. Grandma is using the cell phone to talk to her friend and share photographs of her grandchildren. These two scenarios may each execute on the same cell phone — there is no reason a businesswoman can’t take her grandchildren on a business trip and expect the cell phone to respond to both situations.

Scenario programs, shown in Figure 3, form applications from application elements and define strategies for meeting a mix of system-level I/O demands as multiple application groupings are eligible to execute concurrently. Scenario programs permit decisions about how application elements are scheduled in the context of system-level timing, data, and performance requirements as multiple applications execute on the system.

For example, one valid scenario program might allow the system to be eligible to utilize available resources to process other applications, but prioritize making_a_phone_call. If the user of the cell phone system wishes to send a photograph to a friend at the same time a call is being made, the scenario program would form the applications from the appropriate application elements. In this case, JPEG_encode would feed into rijndael_encryption, which would go through a generation of CRC code to create the sending_pictures application (Figure 4). The scenario program does not need to be limited to simple definition of data dependencies between elements in the style of a task graph, but may include control flow utilizing the system I/O, application deadlines, and task priorities. The Hyperprocessor [13] uses a notion similar to scenario programs to organize tasks.

Another important function of the scenario program is providing system-level data to coordinating programs, such as task dependencies, deadlines, constraints, and expected runtimes. Much like compilers for conventional ISAs, differences in the tuning of the scenario and coordinating programs to the underlying architecture greatly impact overall performance.

Figure 5 is a further illustration of the airport scenario and motivates how scenario programs are developed. A businessman at an airport is receiving some product photos from an associate and browsing_the_web for related information. At a certain point in the

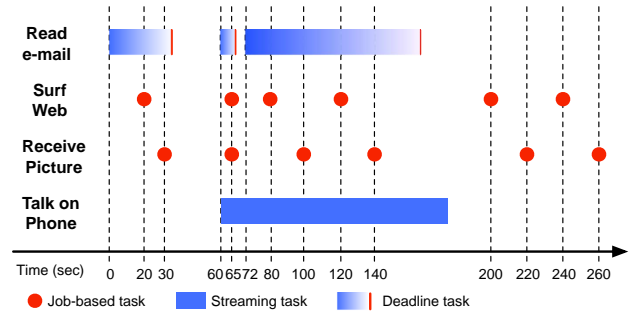


Figure 5 Airport Scenario Timeline

scenario, the businessman will call his associate and have the cell phone read_e-mail_aloud over the phone connection.

Figure 5 is a timeline of baseline cases of response times and overlapping execution patterns of all of the applications (and application elements) that can potentially concurrently execute in the airport scenario. The scenario captures data-dependent execution, fixed response times for streaming (periodic, continuous) inputs, aperiodic arrival times of events, and so the potential to evaluate how well the system responds when one application may still be executing while another starts.

For example, streaming applications, such as the phone call application, are shown in Figure 5 as a solid box indicating their duration. The read_e-mail application, on the other hand, must process the e-mail at a certain rate of words per minute. Although the application may finish sooner and store yet unspoken speech, it does have a soft deadline (shown with vertical lines) in order to meet the lowest speech rate. Finally, job-based applications such as browse_the_web have only their start times shown with dots.

While this particular scenario uses only 4 of the 6 possible cell phone applications, it represents just one of a group of scenarios that may overlap or be sequentially executed on a given PHM. A complete evaluation of a PHM such as this one may include tens of different test scenarios, executing on the same architecture.

4. Cell Phone Example

We use the set of cell phone applications defined in Section 3 to illustrate the usefulness of scenario-based benchmarking to a designer selecting among several different architectures. For the phone call application, we only model the digital parts of the system. The streaming_mp3 audio application streams at 128 kbits/sec and assumes that the network can provide appropriate input bandwidth. Reading_e-mail_aloud uses a speech synthesizer to read already downloaded e-mail to the user. Browsing_the_web ignores the typesetting overhead of the web browser and focuses on decoding multiple small images found on webpages. The send/receive_pictures application involves either encoding an image taken with the phone’s camera, or decoding a received image for display. Unlike browsing_the_web, images processed by this application are typically much larger (>1M pixels).

We focus on three architectures, differing in the number and types of processors. Gross processing power is increased as processors are added, but net processing power is roughly equal.

- *Arch #1:* A single ARM processor running at 350 MHz, including a floating point unit. All applications are executed in round-robin fashion.
- *Arch #2:* Three ARM processors with local memories connected via a bus. The sum of processor clock speeds equals to 400 MHz. Applications are statically scheduled.
- *Arch #3:* Five ARM processors with local memories connected via a bus, adding up to 450MHz. Applications are scheduled onto resources using a dynamic, priority-based scheduler.

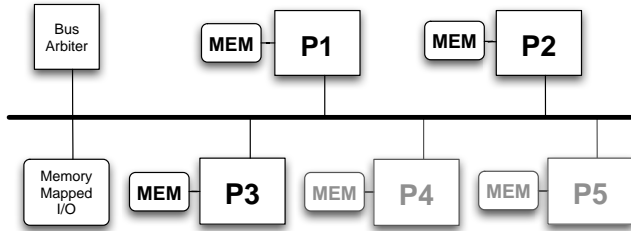


Figure 6 Multiprocessor Architectures

Figure 6 shows architectures #2 and #3; these are heterogeneous due to the differing speeds of the ARM processors and floating point hardware. The bus is a 16 bit wide, 10 MHz bus using a simple lottery-based arbitration strategy outlined in [14] and is implemented with a hardware arbiter. Since every processor has a local memory, the bus interconnect is used to transfer the entire data working set into the processor’s local memory before processing can begin. Similarly, every finished task must use the bus to transfer the output data from its local memory to the memory mapped I/O on the next processor that needs it.

The processors in architecture #2 and its application mappings are listed in Table 1. Notice that the read_e-mail application, which includes a floating point rsynth benchmark, is placed onto P1. Also, MP3 decoding is a more computationally intensive application, and was therefore placed on the most powerful processor. Table 1 also shows the processors within architecture #3. However, the application mappings are not shown because every task is scheduled at runtime onto the best available processor. In the case multiple tasks need to be scheduled at the same time, the scheduler gives priority to streaming tasks such as talk_on_the_phone, stream_MP3, or read_e-mail. This type of scheduling strategy incurs an overhead because the scheduling strategy must periodically be executed on one of the processors.

4.1 Modeling and Assumptions

The cell phone system was modeled using the MESH (Modeling Environment for Software and Hardware) simulator [15]. The MESH simulator permits modeling and simulation of heterogeneous multiprocessor systems at an abstraction level above instruction set simulation (ISS). It determines the system performance by executing application code instrumented by the designer to contain information about application complexity and content. This information is converted into timing at simulation time by taking into account the computational power of the processor(s) executing the type of application functionality, thus permitting a model of the difference in performance as the same task executes on heterogeneous processor resources. The advantage of this approach is that the control and data flow of applications is simulated, yet the simulation speed is significantly improved because individual instructions are not emulated. To gain this advantage, however, the designer must provide some information about application complexity relative to the resource upon which it executes. We ran the benchmarks through the

Table 1 Architecture Detail

μ P	Arch #2	Arch #2 Mapping	Arch #3
P1	200 MHz ARM /w FP	Stream MP3 Read e-mail	200 MHz ARM /w FP
P2	100 MHz ARM	Browse web Receive pictures	100 MHz ARM /w FP
P3	100 MHz ARM	Phone call Send pictures	60 MHz ARM
P4	--	--	60 MHz ARM
P5	--	--	30 MHz ARM

ARMulator ISS (included in the gdb distribution [16]) in order to gain this information. The annotation points were carefully chosen to ensure annotation validity for any size or content of input data, thus data-dependent execution times are modeled above the ISS-level. We found that the MESH performance estimation was within 10% of the ARMulator for all applications, with most applications being accurate within 1%. However, because the MESH executes at much higher abstraction level, it is two orders of magnitude faster than the ARMulator for most benchmarks.

The bus interconnect was also modeled above the cycle accurate level. Since the bus is designed to primarily move large continuous amounts of data between the processors and the memory mapped I/O, it mainly operates within a burst mode. The burst mode allows a processor to gain bus access for a multiple number of cycles, allowing the arbitration for the bus to occur much less frequently. Our simulations perform bus arbitration every 100 bus cycles.

The bus arbitration strategy uses tickets issued to various masters to ensure fair arbitration. As such, the highest number of tickets is issued to streaming applications, making them more likely to gain the bus in the case of contention. Additionally, we differentiate the phone call application as more important than other streaming applications.

The execution overhead of the scheduling strategy used in architecture #3 is modeled by applying an overhead of 2000 processor cycles per each task-to-processor pairing decision. This is a conservative estimate, considering the simplicity of the scheduling algorithm and the numbers of processors and tasks used in this example. The scheduling strategy must also gain control of the system bus to identify which processors are idle prior to scheduling, as well as to disseminate the results after scheduling, presenting additional overhead.

4.2 Results

We use the airport scenario introduced in Section 3 to motivate some of the design decisions that can come from scenario-based benchmarking. We use the response time of job-based applications, such as browsing_the_web and receiving_pictures, as a metric for evaluation of the three architectures. Figure 7 shows every instance of a job-based application on its x-axis and their response times on the y-axis. Each group of bars corresponds to a dot in Figure 5. (Also executing are the scenario’s streaming applications.)

Consider the response times for tasks starting after $t=200$ sec. This represents the performance of the architectures when the benchmarks execute one at a time, as for conventional benchmarking. Informed only by this, a designer would choose the single processor, since it outperforms the multiprocessor (MP) architectures and presents less of a design challenge. For this example, when running one or two applications at a time, it is hard to argue against a single processor design.

However, when the system is heavily loaded (as between $t=65$ sec and $t=140$ sec), the execution of the read_e-mail and phone_call applications in the background increase the architecture #1 response time of browse_the_web and receive_pictures applications significantly. However, the higher cumulative computational power of the multiprocessor systems in architectures #2 and #3, provides better results. Using individual benchmarks, the designer cannot evaluate the interference various concurrent applications may have with each other. Randomly running multiple applications at a time would be deceiving as well; some applications make sense to run concurrently and some do not. In the end, the best evaluation of the system results from running relevant scenarios that are likely to occur during the normal usage of the system. In this example, the designer might decide that the events between $t=65$ and $t=140$ are highly unlikely and optimize the system for low usage loads.

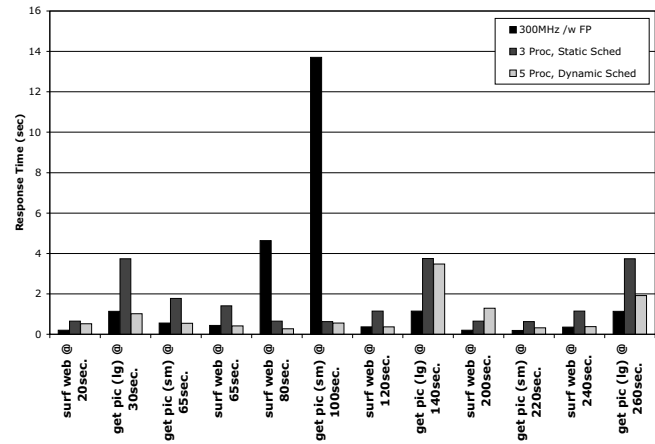
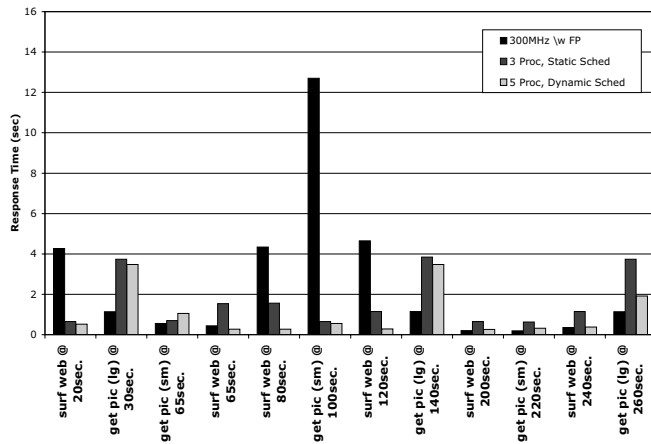


Figure 7 Airport Scenario Results for Normal e-mail (left) and “Green Eggs and Ham” (right)

Similarly, the data input into the system can have a significant impact. For example, let us assume that instead of reading business e-mails, the read_e-mail_aloud application instead is reading excerpts of identical length from “Green Eggs and Ham,” a children’s book by Dr. Seuss. Otherwise, the businessman at an airport scenario remains the same.

Figure 7 (right) shows that reducing the word complexity in the data input for the read_e-mail application reduces the load on the system and is most notably visible in architecture #1. Although containing the same number of words, e-mails containing “Green Eggs and Ham” are much simpler for speech synthesis, allowing the read_e-mail deadline tasks starting at $t=0$ and $t=72$ to complete sooner. As a consequence, surf_the_web tasks at $t=20$ and $t=120$ have much improved response times for the single processor case because more processor power is available.

It is easy to see that other inputs to the scenario may impact the performance of the system. For example, moving some of the task deadlines earlier requires the deadline task to consume more processing power in a shorter period of time, affecting the other concurrent tasks on the same resource. Because PHM designs are sensitive to much more than input data and applications in the system, scenario-based benchmarking considers other important system inputs such as task arrival times, data dependencies, concurrency and deadlines.

5. Conclusions

We introduced scenario-based benchmarking, or *scenarios*, a new vision for modeling and evaluating single chip programmable heterogeneous multiprocessor designs. Scenarios are a design methodology oriented towards helping designers pinpoint potential bottlenecks or limitations of hardware and system level software, early in the design cycle.

Through an example illustration, we showed that PHM designs cannot be evaluated by simply running benchmarks appropriate for single processors as has been done in the Computer Architecture community. Similarly, we concluded that, unlike the Design Automation community, concurrently executing timed and untimed software applications must be considered as application groupings in order to accurately represent the performance of a PHM. Thus, scenarios which include application elements, I/O datasets, and scenario programs, must be considered.

Our vision and the results of this paper serve to motivate several new research directions. The current work only touched on the importance of scenario-based benchmarks. But even the SPEC

benchmarks took years to develop; considerable work remains in developing and classifying scenarios for the PHM design space.

6. Acknowledgements

The authors would like to thank Alain Mellan of ST Microelectronics and Brett Meyer and Josh Pieper of the MESH team and for their suggestions. 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] Special Issue 2002 on Ubiquitous Computing in *Communications of the ACM*.
- [2] W. Wolf. “How many system architectures?” *Computer*, March 2003, pp. 93 -95.
- [3] International Technology Roadmap for Semiconductors 2003 <http://public.itrs.net/>
- [4] J. Paul. “Programmers’ Views of SoCs,” *CODES-ISSS*, 2003.
- [5] www.openmp.org
- [6] K. Keutzer, et. al. “System-Level Design: Orthogonalization of Concerns and Platform-Based Design,” *IEEE TCAD*, p. 1523-1543, Dec. 2000.
- [7] <http://www.simplescalar.com>
- [8] <http://www.spec.org>
- [9] <http://www.eembc.com>
- [10] <http://www.bdti.com>
- [11] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. “MiBench: A free, commercially representative embedded benchmark suite.” *Proc. 4th Workshop on Workload Characterization*. pp. 1-12. 2001.
- [12] C. Lee, M. Potkonjak, and W. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. *ISCA 1997*. pp. 330-335.
- [13] F. Karim, A. Mellan, B. Stramm, T. Abdelrahman, U. Aydonat. “The Hyperprocessor: a Template Architecture for Embedded Multimedia Applications.” *WASP 2003*.
- [14] K. Lahiri, K., A. Raghunathan, A., et. al. “LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs. *DAC 2001*.
- [15] A. Bobrek, J. Pieper, J. Nelson, J. Paul, D. Thomas, “Modeling Shared Resource Contention Using a Hybrid Simulation/Analytical Approach,” *DATE*, 2004.
- [16] www.gnu.org/software/gdb/