Power-Performance Simulation and Design Strategies for Single-Chip Heterogeneous Multiprocessors

Brett H. Meyer, *Student Member*, *IEEE*, Joshua J. Pieper, JoAnn M. Paul, *Member*, *IEEE*, Jeffrey E. Nelson, *Student Member*, *IEEE*, Sean M. Pieper, and Anthony G. Rowe

Abstract—Single chip heterogeneous multiprocessors (SCHMs) are becoming more commonplace, especially in portable devices where reduced energy consumption is a priority. The use of coordinated collections of processors which are simpler or which execute at lower clock frequencies is widely recognized as a means of reducing power while maintaining latency and throughput. A primary limitation of using this approach to reduce power at the system level has been the time to develop and simulate models of many processors at the instruction set simulator level. High-level models, simulators, and design strategies for SCHMs are required to enable designers to think in terms of collections of cooperating, heterogeneous processors in order to reduce power. Toward this end, this paper has two contributions. The first is to extend a unique, preexisting high-level performance simulator, the Modeling Environment for Software and Hardware (MESH), to include power annotations. MESH can be thought of as a thread-level simulator instead of an instruction-level simulator. Thus, the problem is to understand how power might be calibrated and annotated with program fragments instead of at the instruction level. Program fragments are finer-grained than threads and coarser-grained than instructions. Our experimentation found that compilers produce instruction patterns that allow power to be annotated at this level using a single number over all compiler-generated fragments executing on a processor. Since energy is power*time, this makes system runtime (i.e., performance) the dominant factor to be dynamically calculated at this level of simulation. The second contribution arises from the observation that high-level modeling is most beneficial when it opens up new possibilities for organizing designs. Thus, we introduce a design strategy, enabled by the high-level performance power-simulation, which we refer to as spatial voltage scaling. The strategy both reduces overall system power consumption and improves performance in our example. The design space for this design strategy could not be explored without high-level SCHM power-performance simulation.

Index Terms—System architectures, integration and modeling, power management, low-power design, energy-aware systems, performance analysis, design aids.

1 Introduction

Single chip heterogeneous multiprocessors (SCHMs) are becoming more commonplace for embedded and semi-custom portable devices. The complexity and variety of SCHM architectures will likely increase as no single standard platform has emerged [1], [2]. Multiprocessor frameworks like the Hyperprocessor [3] motivate how collections of heterogeneous processors might be organized, but also leave a large design space to be explored—and exploited. There are two primary reasons for this. First, it will soon be possible to place a hundred ARM-equivalent processors on single chips. The selection of numbers and types of processing elements (PEs) is not straightforward. Second, the applications these systems will execute will have unprecedented forms of heterogeneous parallelism that can be exploited in new ways.

Applications that previously used to be considered in isolation are beginning to be integrated for simultaneous execution on the same computing device, creating entirely new application sets. The merging of increasingly sophisticated Human Computer Interface (HCI), Computer-Computer Interface (Networking), and Environment-Computer Interface (Classic Embedded or real-time applications), along with many of the applications previously associated with desktop computers, will form new classes of applications. A common example of this kind of system is the cell phone, which can also serve as a Web browser and, with more sophisticated HCI, a personal digital assistant (PDA), executing functionality previously associated with laptops or even desktop computers.

Models, simulators, and design methodologies that target the instruction-level of individual PEs (or below) will be inadequate to capture the rich set of design tradeoffs required in this new design space. This is true not only because of prohibitively large simulation times when instruction set simulator-level (ISS) simulations have even a few PEs, but because of the time to develop each ISS-level model. The design of these next generation SCHMs is more appropriate at the thread level instead of at the instruction level, i.e., when the unit of execution is thought of as a

- *B.H. Meyer, J.M. Paul, J.E. Nelson, and A.G. Rowe are with the Electrical and Computer Engineering Department, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. E-mail: {bhm, jpaul, jnelson, agr}@ece.cmu.edu.*
- *J.J. Pieper can be reached at 129 Franklin St., Apt. #421, Cambridge, MA 02139. E-mail: jjp@pobox.com.*
- *S.M. Pieper is with the Electrical and Computer Engineering Department, University of Wisconsin-Madison, 2414 Engineering Hall, 1415 Engineering Dr., Madison, WI 53706-1691. E-mail: spieper@wisc.edu.*

Manuscript received 27 Feb. 2004; revised 2 Aug. 2004; accepted 8 Oct. 2004; published online 15 Apr. 2005.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TCSI-0067-0204.

thread instead of an instruction. (Here, we use thread as a fundamental unit of concurrency most analogous to a superinstruction since it is a sequence of instructions that has the potential to be scheduled and also share common state with other threads.) Beyond this, registers, cycles, and instructional units must be elevated to new design elements appropriate for this level of design. New design elements for physical resources and functional execution are required, along with new relationships between them. Further, execution speed can no longer be distilled to a single number for worst-case or average behavior, but will result from trade-offs between application sets that simultaneously execute in the system in a variety of situations [4].

A primary motivation for designing SCHMs is to exploit new design strategies at the system level to save power. There are two well-known examples for this: substituting multiple PEs at lower clock rates for one at a higher clock rate and turning off PEs when they are not needed. Both of these approaches may become even more important as leakage current begins to dominate power consumption and, instead of reducing transitions, the goal is to reduce the number of unnecessary, powered-on transistors. Here, the use of collections of smaller, simpler cooperative PEs executing at lower frequencies is attractive for three reasons. First, transistors that support the performance for general purpose computation are eliminated as processor types can be more suited to application types. Second, the cost of multiplexing many unrelated or loosely related threads on a high-speed processor can be eliminated by instead executing the threads in parallel. Third, the intelligent management of collections of processors that can satisfy peak loading, but be turned off otherwise, can satisfy different levels of demand on the system while reducing overall power consumption.

The discovery of new strategies for exploiting this new design space will be enabled only if performance and power can be meaningfully and accurately modeled, manipulated, and simulated at a new level of design. Toward this end, this paper has two main contributions. First, we start from the basis of an existing high-level performance simulator and investigate how to extend it into a power-performance simulator. MESH (Modeling Environment for Software and Hardware) can be thought of as a thread-level simulator for an SCHM instead of an instruction-level simulator for a microarchitecture. Using MESH, we have had success exploring SCHM design well above the level of the ISS, where designers manipulate threads, processors, scheduling, and communications strategies instead of instructions, functional units, and registers. We have compared our performance results with ISS-level models and found reasonable accuracy while execution is typically two orders of magnitude faster than ISS-level models. However, in extending MESH to model power as well as performance, we found a lack of consensus for how MESH threads might be annotated for power modeling. Our hypothesis was the intuitive assumption that different program fragments, which are finer grained than threads, would have different instruction mixes and, so, require the annotation of different power consumption values in the MESH threads. Thus, we sought a method for identifying different fragment types and calibrating the high-level model to their individual power consumptions. We ran experiments over a subset of the SPEC CPU2000 [5] and MiBench [6] benchmarks. Across a wide variety of programs types, we found that compilers tend to produce patterns of instructions that generate a mix of both instruction sequences and instruction types. This allows a single, average power consumption, calibrated once to each PE type, to sufficiently model the execution of all compiler-generated program fragments that execute on that processor type. The exception is when designers generate extraordinary instruction mixes by hand, such as when a processor is in an idle state. But, in these cases, threads are performance-tuned, or even power-tuned, such that designers can readily calibrate power consumption at the time the thread is designed as a one-time measurement. Since energy is power*time, this simplification of power modeling leads to system runtime (i.e., performance) being the dominant factor to be dynamically calculated at this level of simulation. Our experimental results also suggest that the power characteristics of interprocessor coordination and interactions, in the form of scheduling, memory hierarchies, and on-chip network communications, will likely be far more important than the power consumption of instructions or instruction sequences of individual processor cores in an SCHM.

High-level modeling is most beneficial when it opens up new possibilities for organizing designs. Thus, our second contribution is a new design strategy enabled by the high-level power-performance simulation. We refer to this as *spatial voltage scaling*, one design strategy of many that can be enabled by providing power-performance simulation at the new level of design provided by MESH. Spatial voltage scaling reduces both overall system power consumption and improves performance over a baseline design in our example. The design space for this strategy could not be explored without high-level SCHM power-performance simulation. Not only would the simulation time have been prohibitive, an additional barrier to creative design organization exists when designers are thinking in terms of instructions, registers, and cycles instead of threads, processors, and synchronization. Thus, the two contributions are related.

2 Design Space Complexity

Future SCHMs will be design hybrids, containing characteristics of processor design as well as hardware design [7]. Models and simulators that support the exploration of the new design space of SCHMs must capture essential features while eliminating unnecessary detail. The central question in the support of high-level design is always what detail can be eliminated and what is essential. The examination of this for power annotation is one of the contributions of this paper. In order to understand the type and amount of detail required in any new model, the design space must first be understood.

For future SCHMs, the design space arises from at least four distinct parts: the application, the testbench, the architecture, and the schedulers. None of these need to be given up front, but the amount that is known in advance about each part can have a large impact on how to strategize design exploration. For example, the broader the application space, the more the solution requires architectures and schedulers that handle a variety of situations well. Many future SCHMs will have more

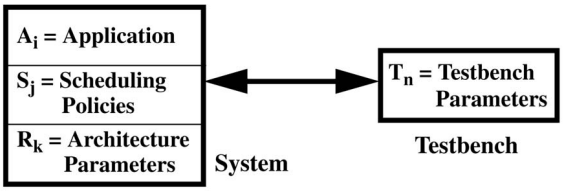


Fig. 1. A point in the design space.

application specificity than general purpose processors, but less application specificity than pure hardware design.

At a high level, the design space may be broken into four variables: system architecture configuration (R), system scheduling policy (S), system application (A), and testbench composition (T). In general, R, S, A, and T are vectors which may contain either vectors of numerical parameters representing configurations, specifications of physical system elements, or full software programs. This constitutes a rich set of design possibilities. A point in this layered design space is shown in Fig. 1.

Even when the application set is known in advance, the remaining design space is large. The testbench merely models the inputs that exercise the system, but the data values, data sizes, and arrival rates of these inputs can prompt a variety of interactions in SCHMs. Thus, T may be amenable to parameterization in terms of data content, data size, and data arrival times, but it cannot be parameterized into a finite set of cases without some knowledge of the system it is exercising. Further, T itself is a vector since it contains both data and time values. Unlike traditional benchmarking, where data is an untimed input to the system, future portable devices will be evaluated against a variety of timed input sets [7], more like the testbench of a hardware design language (HDL) such as Verilog or VHDL.

Each instance of the system architecture R can be thought of as a vector which includes

$$R = \{\text{numbers of processors, processor types, frequency of each processor, method of interprocessor communication}\}.$$

While the specification of the physical architecture is clearly important to any design space exploration, an equally important aspect of the design is the system scheduling algorithms; when collections of processors are used to solve a common problem (or set of problems), they must cooperate in an intelligent manner and schedulers are responsible for ensuring this happens. Scheduling is often implicit when single processors are used to execute applications; the scheduling is contained right in the program. When exploiting thread-level parallelism over a collection of processor resources, however, cooperative scheduling must often be developed in conjunction with the application and the features of the underlying architecture and possibly anticipating the way the system will be exercised. Further, the cost of cooperation must be evaluated. Interprocessor coordination is not free, but comes at the price of the overhead to support a common scheduling or communications domain.

Since schedulers, like other software programs, are designable algorithms that execute in the final design, S can represent an infinite set of programs. While some scheduling parameters can be specified—for example, how long a scheduler waits before shutting down a processor—the design of programs that make data-dependent decisions cannot be distilled to numerical sets of values generated by other programmatic, analytical models.

This blend of software and hardware and the level of design complexity result in the need for mixed software and hardware simulation above the ISS-level. We have been developing the basis for a high-level simulator to capture the performance trade-offs in this design space, which we discuss in the next section. This will be the basis from which we will approach high-level power-performance modeling.

3 MESH

This section provides an overview of MESH as a preexisting performance simulator that permits the designer to efficiently explore the design space described in Section 2, at the thread-level as opposed to the instruction-level. MESH has been previously described and evaluated [7], [8], [9], [10], [11]. In MESH, system performance is simulated by resolving software execution into physical timing using high-level models of processor capabilities, with thread and message sequence determined by schedulers. The central question to be asked of all simulators is what is in control of sequencing the system. By interleaving heterogeneous PEs on the basis of physical time intervals, we capture the physical timing basis all performance simulators require without resorting to a global simulation tick. However, software does not contain physical timing properties, but, rather, coordinates state update logically, with performance determined only by the relative computational capabilities of the underlying resources upon which it executes. Thus, the most significant feature of MESH is that the amount of computation complexity advanced by a given PE in its execution interval is calculated at simulation time according to the amount and form of the software eligible to execute on that PE. Performance modeling of the entire SCHM is, in turn, calculated by how logical state is coordinated as it is advanced by a collection of PEs. Thus, MESH captures the different sequencing of software and hardware, resolved though scheduling and communication with new design elements. These elements are based upon thread types which capture the contributions to overall system performance of both logical and physical sequencing [12] as well as the layered resolution of the two fundamental kinds of sequencing, also captured as thread-based design elements.

Fig. 2 illustrates the primitive modeling elements of MESH at a high level. This view provides the essential modeling elements of design of SCHMs at the thread level as opposed to the instruction or cycle level. At the lower left of the figure, an SCHM is illustrated interacting with some external testbench. The SCHM includes n PEs, with n separate clocks. The cooperation of the PEs in the SCHM is illustrated as a layer of "scheduling and arbitration," shown as a solid, thick line. Many overlapping layers of scheduling and arbitration may exist in SCHMs; the single domain is shown to simplify the illustration of the main design elements of MESH. Also, the label "scheduling and arbitration" may represent networked communications as well as shared buses; the label in the figure simply illustrates processors grouped into a logical, cooperative domain.

In Fig. 2, $PE1$ is expanded into a MESH "thread stack." The thread stack illustrates all of the design elements contributed by $PE1$ (each PE contributes a similar thread

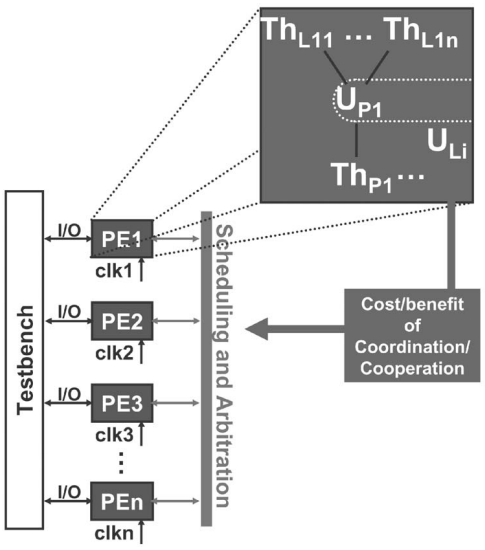


Fig. 2. Layered logical and physical design elements in MESH.

stack to the SCHM). At the bottom is a model of the physical capabilities of the processor resource, modeled as physical thread Th_{P1}. Each unique PE represents a different physical capability within the SCHM, which can be programmed to carry out concurrent behavior. This programming is captured as a collection of logical threads, $Th_{L11}, \ldots, Th_{L1n}$, shown at the top of the expanded view. The logical threads represent the untimed, logical sequencing of software. This simply captures the effects of software executing on hardware, where physical timing is not determined until the software executes on a model of or a real physical machine. Each PE may execute an unbounded number of software (logical) threads, with scheduling decisions potentially made dynamically in response to different data sets. This per-processor, thread-level decision making is modeled by a physical scheduler, U_{P1}.

Taken alone, this might represent a main program or OS executing on a single processor resource. However, the key to the design of SCHMs is the design of cooperation among a variety of PEs. The cooperation of the various PEs as state is exchanged and scheduling decisions are made is modeled by the cooperative, or logical scheduling domain, U_{L1}. Because U_{L1} represents a cooperative scheduling domain, the threads that execute on any given resource may also be eligible to execute on any other resource that belongs to the same logical scheduling domain. This is true even if the processors are heterogeneous (we will utilize this in our example in Section 7).

It is significant that each logical thread at the top of the diagram need not have a one-to-one correspondence to the underlying hardware elements. It may be desirable for some threads to be mapped to processor resources and even be guaranteed to be available to always execute on them, while other groups of threads can execute on groups of possibly heterogeneous processor resources and, so, are considered mapped to a logical scheduling domain instead. In either case, the number of logical threads may even be unknown until the system is executing, responding to different data sets.

Because U_{L1} captures the penalty of resource sharing and cooperation [11], we can model what is perhaps the key design challenge of the design of SCHMs at a greatly reduced level of modeling and simulation detail—the trade-off between a few powerful, complex, PEs that execute more threads locally or more less-powerful, simple, PEs with the additional cost of global cooperation.

The middle scheduling layer (U_{P1}, which executes on $PE1$ and contributed to the common scheduling domain U_{Li}) serves an additional purpose in the simulator. MESH's schedulers and resource models enable performance modeling by resolving the different timing of software and hardware—logical and physical timing. The key here is that the schedulers and logical threads use *consume calls* to resolve the logical computation of the software threads to the physical resource power provided by the resource thread. Consume calls are tuples of information, representing the amount of software complexity within a program fragment. We define a *program fragment*, which we will sometimes refer to in this paper as a *fragment*, as the granularity of performance annotation (speed or power) in a simulation. Fragments may be as large as the thread-level or as fine as the instruction-level, but, in general, are finer-grained than individual threads and coarser-grained than individual instructions. If fragments are too coarse-grained, accuracy suffers. If fragments are too fine-grained, simulation performance and, potentially, design time suffer. However, the important point is that the annotation of consume values within threads is flexible and an important part of creating a good model.

Another important point about consume calls is that they represent timing that is relative to the physical capabilities of the resource upon which the fragment executes. As a simple example, an annotation of "consume(9)" indicates that nine units of computational complexity have been consumed by the fragment since the preceding consume call. However, the nine units of complexity may result in a different amount of actual execution time on different physical (processor) resources. Given a more powerful processor or a processor that was capable of executing functionality unique to the logical thread (such as if the processor had a floating-point unit and the logical thread contained many floating-point instructions), it would execute more logical events per activation, giving a different state trajectory to the system. Thus, processor heterogeneity is modeled as threads execute on different resources (scheduled both statically and dynamically).

The basic types of threads in MESH are summarized as:

- Th_{Lij}—One of j logical threads (software) that will execute on processor i.
- Th_{Pi}—A model of the ith physical resource in the system, such as a processor.
- U_{Pi}—A scheduler that selects logical threads intended to execute on resource Th_{Pi}.
- U_{Li}—A logical scheduler that can schedule M threads to N resources.

We have done prior experimentation with how to annotate software for reasonable accuracy at high-level modeling. MESH has been previously shown to enable designers to evaluate the performance effects of design trade-offs in the software, hardware (numbers and types of PEs), scheduling decisions, and communications (memory arbitration and network-style protocols) across multiple PEs on a chip, with accuracy typically within 10 percent of ISS-level simulations,

but which execute two orders of magnitude faster [9], [10]. We will now examine the necessary granularity of power annotation required in MESH. This will form the basis of the first contribution of this paper.

4 Power-Performance Modeling Prior Work

We begin our investigation of how to include power modeling in MESH with an overview of prior work in power modeling and simulation. We will ultimately use this to draw a set of parallels between instruction-level power modeling and thread-level power modeling.

Analytical power models rely on known worst-case execution times, averages, or statistical properties to quickly determine power consumption in a system which can be completely parameterized [13], [14], [15], [16], where programmatic decision making does not affect overall performance. Because of the need to model schedulers as programs, analytical models remove too much detail for our purposes.

At a much lower, microarchitectural level ISS models and even RTL-level models can inherently capture data-dependent and processor-specific execution unless the design space is otherwise restricted [17]. The cost of models at this level is in the effort required to develop them and the time required to execute them. Projects like Wattch [18], SimplePower [19], and Sim-Panalyzer [20] are frameworks for evaluating trade-offs in microarchitecture power dissipation and performance. All three are based on the SimpleScalar microarchitecture simulation platform [21] and each performs power estimation by combining device capacitance models with measures of switching activity. The microarchitecture-level focuses on simulating the affects of minor functional changes on performance; they are also becoming the reference models against which high-level models are evaluated in lieu of actual hardware.

Our objective lies somewhere in between analytical models and ISS-level models and, thus, we will pay the most attention to prior work in this area. These models estimate power based on information about instructions or whole applications a priori. We are most interested in these techniques to inform power modeling in MESH since they estimate the power of dynamic instruction sequences without actually simulating the instruction sequence.

We observe that there are four fundamental approaches to power estimation between analytical models and ISS-level models, which we will number for later use in the paper:

A1. *Using a single power value to model all instructions in a given processor*. This approach has been proposed to be sufficient when it can be assumed that there is little interinstruction dependency and variations in the energy per cycle cost of instructions do not coincide with the variation in instruction mix across programs. Prior work here is exemplified by [22], in which measurements were taken of instruction and inter-instruction costs on a RISC machine. Based on these measurements, the conclusion was drawn that, with a single power value, we can model the overall power consumption of a processor with 8 percent error.

A2. *Measuring classes for groups of instructions (without regard to underlying structure)*. In this approach, after exhaustive measurements are taken, instructions with similar energy cost per cycle are then grouped to reduce the number of power values needed to characterize the system [23], [24]. For example, instruction clustering is shown to model power for a VLIW processor with less than 2 percent error in [23]. This method is certain to reduce the error inherent in using a single power value to characterize a processor.

A3. *Assigning power values to structural features related to the underlying architecture*. In this approach, knowledge of the underlying processor architecture informs the estimated cost of instructions. In [13], a processor is divided into functionalities, or independent operations, like Fetch and Decode, Branch, Load and Store, ALU, for example. Once the cost of each functionality is determined, the cost of an instruction is derived from the cost of the functionalities it requires for execution. This information can be easily used to group instructions that all use the same set of architectural elements.

A4. *Using context to inform power modeling*. This approach is based on the presumption that the power consumption of instructions executed in sequence may be different from that of the instructions considered (and measured) individually. The basic idea of [25] is to measure the average power consumption of each instruction and instruction pair in the target processor (in this case, a DSP) and then apply this information to determine the total cost of an application. This method modeled power in the DSP with 11 percent error. This method requires the evaluation of each instruction pair, including combinations of addressing modes.

Based upon this set of prior work, we were faced with choosing which method would be most appropriate for annotating the power consumption of program fragments in MESH. Since there is no clear consensus, we performed our own set of experiments to determine which method should be used and to guide future work so that power may be modeled at the new level of design provided by MESH.

5 Experiments for High-Level Power Annotation in MESH

Our objective was to understand how to obtain the individual power values that should be assigned to individual program fragments in MESH under the hypothesis that different fragment types would lead to different power values and, possibly, different techniques for assigning power values. We began with the goal of identifying which fragments exhibited different average power consumption so that we would be able to classify fragments as having different power signatures (or just "signatures"). A power signature is an energy per cycle value in effect while a given fragment is executing on a given processor. Thus, if the execution time of a fragment with a certain power signature is calculated by the simulation, its energy cost can be easily calculated by the simulation.

For our experimentation, we developed a custom-design board and used an ISS-level power simulator. These were intended to obtain calibration values for the power modeling on a per-fragment basis, as well as to help us to obtain measurements on a per-fragment basis over a variety

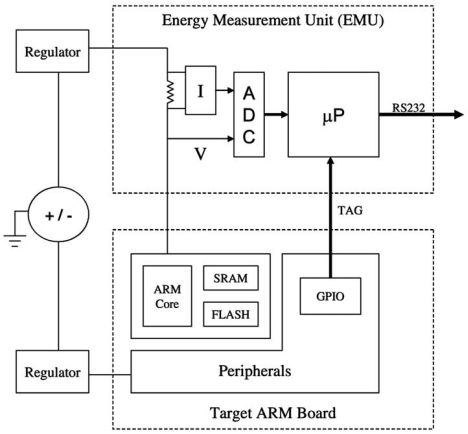


Fig. 3. Architecture of energy measurement unit (EMU).

of fragments so that we could draw some conclusions about fragment type and power annotation. In this section, we describe the experimental set-up, our experiments, and our experimental results for high-level power calibration and annotation.

5.1 Experimental Setup

We used Sim-Panalyzer v.2.0 [20], with the default configuration for the ISS-level simulator. This includes the use of default power consumption characteristics, as well as the default memory model, issue width, and instruction window. The simulator was necessary in order to provide fine-grained control and monitoring of instruction execution. However, because many ISS-level simulators must also be calibrated to actual architectures, we also obtain some of our energy measurements through the use of the Energy Measurement Unit (EMU), a custom device that permits power measurement on an actual processor.

The architecture of the EMU is shown in Fig. 3. The target ARM board is a Phillips LPC2106, an ARM7TDMI-S-based development board. The ARM has 128 kB of on-chip FLASH and 64 kB of on-chip SRAM. In our experiments, test binaries were uploaded to the FLASH and then executed from the FLASH. The LPC2106 has a memory acceleration module that performs prefetching to ensure that one instruction is available for issue each clock cycle.

The EMU offers several advantages over multimeters (used in [27], [28], [29] among others) for microprocessor energy measurement and avoids some methodological errors common in previous work. Unlike previous work, which assumed a constant input voltage to the device under test, the EMU simultaneously measures current and voltage. Assuming that the supply voltage is constant creates a source of error and inflates the measured energy of high-current instructions. By measuring both current and voltage on the output of the power supply, we eliminate a source of variation and can be more certain of relationships such as "*Branch* takes four times the energy of *Add*." As power supplies may leak current and have slow response times, such statements are hard to justify otherwise. The data logs generated by the EMU show fluctuation in the voltage provided to the core, which is inversely related to change in consumed current, validating the need to measure both supply voltage and current.

To isolate the input current from the measurement circuitry, we use a Maxim 4372h current to voltage converter with built-in gain and internal current mirror. This device's 3dB cutoff is near 200KHz. Thus, we sample at only 100 KHz to avoid a significant loss of precision. We observed less than 0.1 percent variation between nearby measurements of identical code segments (more distant measurements show a variation of approximately 1 percent due to thermal differences).

In order to identify code segments on the EMU, we use an 8-bit tag port (output pins driven by a global I/O register on the LPC2106) connected to the system under test. When the EMU detects a change on the port, it transfers the data off-board to a logging program. This transfer requires that the program running on the system under test must pause for approximately 2.8 ms between changes in state.

While the EMU is very precise for code segments that execute over a long period of time, it is not suitable for creating energy profiles of code that rapidly shifts between key segments. It is necessary to use code segments that execute for several thousand cycles in order to amortize the effects of last measurement slack. Therefore, we use the EMU so that we can compare the execution of specific program fragments against both an ISS-level simulator and real hardware. Typically, these fragments were constructed in such a way as to isolate the power consumption of individual instructions.

5.2 Experiments

The first set of experiments measured the cost per cycle of individual instructions. On the LPC2106, instructions were executed 10,000 times in a row, with operands held constant, and the average power calculated using the power monitor described in Section 5.1. On Sim-Panalyzer, instructions were executed 10 and 10,000 times, with the average power calculation resulting from a line fit to these two points. This was done to account for startup and shutdown overhead.

We also ran a variety of benchmarks on Sim-Panalyzer. Together with instruction mix information that was gathered using the GDB/ARMulator, a GNU ARM ISS, these experiments were conducted to test how much instruction mix and the related energy dissipation per cycle varied from application to application. The benchmarks chosen included the applications used in the example in Section 7, selections from MiBench [6], and SPEC CPU2000 [5]. From MiBench, we used *FFT*, *lame*, *jpeg*, *rijndael*, and *rsynth*. From SPEC, we used *crafty*, *vortex*, and *vpr*. Finally, we also ran a custom set of applications that we used later in our example: *gzip*, *wavelet*, and *zerotree*. These benchmarks were all compiled in gcc with optimization O3. The MiBench and custom benchmarks were executed with typical and pathological input data where possible. All SPEC benchmarks were executed using the MinneSPEC workloads [30]. We anticipated that we would be able to bin program fragment by average power consumption, leading to a further narrowing of the problem of deciding when and how to obtain different power annotations for program fragments in MESH.

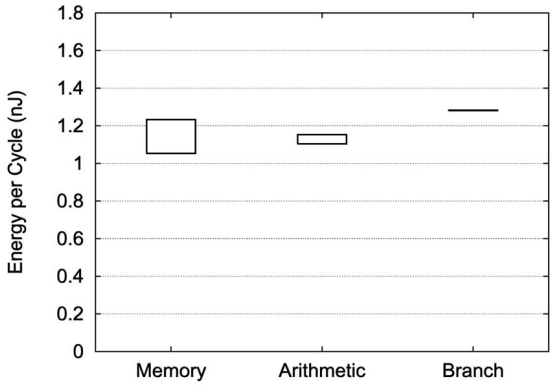


Fig. 4. Instruction cost variation in the LPC2106.

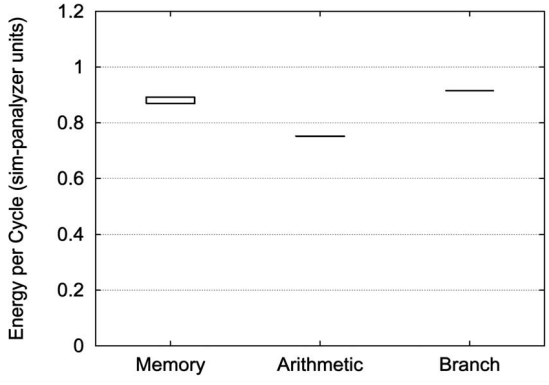


Fig. 5. Instruction cost variation in Sim-Panalyzer.

5.3 Experimental Results for High-Level Power Annotation

We sought to understand how prior work in power modeling between the RT-level and analytical models, as described in Section 4 as approaches A1-A4, might be applied to the high-level annotation of program fragments in MESH. By applying the general form of the A1-A4 approaches, we are also able to point in the direction of how high-level power annotation might be achieved for both processor cores and at the system level.

We discovered that, for compiler-generated tasks, over the wide range of benchmarks we tested, a single power value sufficiently modeled the power consumption of all threads executing on a given processor. Thus, approach A1 in Section 4 holds with reasonable accuracy for compiler-generated code over all of the benchmarks we tested since these were compiler-generated fragments. We found this to be the case because, across instructions, the energy per cycle is relatively invariant, across fragments the instruction distribution is relatively invariant, and the interinstruction and architectural effects are small across all of the benchmarks that we executed.

To see this, first consider the variation in instruction costs in Fig. 4 and Fig. 5. Fig. 4 shows the minimum and maximum energy per cycle cost of several classes of instructions measured by the EMU board. Fig. 5 shows the same results when measured using Sim-Panalyzer. The two results are very similar and show a similarly small variation, with approximately 22 precent difference between the overall minimum and maximum in Fig. 5.

Next, consider the variation in instruction mixes in Fig. 6.

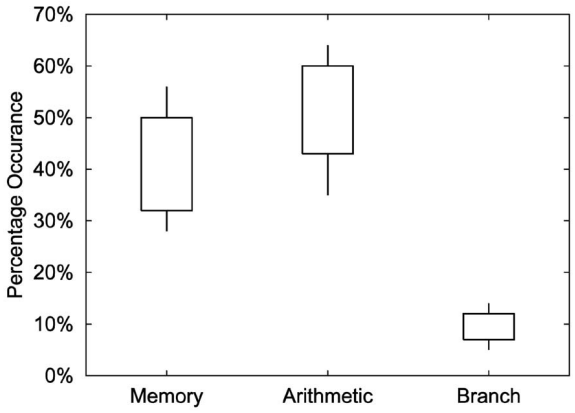


Fig. 6. Instruction mix variation on GDB/ARMulator.

In this figure, 75 percent of the benchmarks fall into the boxes for each instruction class; the lines indicate maximum and minimum percent occurrence of a particular instruction class. The instruction class that showed the largest variation in occurrence, memory operations, changed from 32 percent to 50 percent over the middle 75 percent of all our tested benchmarks. The average energy per execution cycle across all benchmarks is 1.13 units as can be seen in Fig. 7, which shows the processing core energy per cycle cost of each benchmark. Using this average value of 1.13 to model power, the average error for each individual benchmark is 6 percent, while the maximum error is 15 percent. The measurements for each benchmark were taken in Sim-Panalyzer.

We conclude that, when modeling the power consumption of compiler-generated threads executing on processor cores on SCHMs, a single power signature may be used to represent all compiled programs for a given processor with reasonable accuracy. This is based on the observation that a variety of different benchmarks, compiled under *gcc*, vary only slightly in their mix of instructions. With only minor variation in benchmark instruction mix, the individual cost of instructions must differ dramatically to impact the average cost of a program as a whole—and we observed through direct experimentation that they do not. This result is supported by the observation in [35] that a low proportion of system power is dissipated in the actual execution of instructions; the majority of the power is dissipated in actions that occur for each instruction, such as fetch and decode. Additionally, compilation plays a key

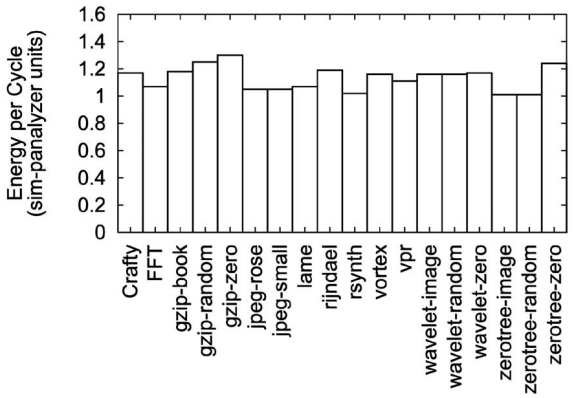


Fig. 7. Benchmark core energy/cycle cost.

role in smoothing out the types of instructions that execute on a processor. In retrospect, this is not surprising since compilers use patterns to generate code. It is unlikely, therefore, that an ARM executing gcc-compiled code would require more power signatures when executing most real applications.

While we did not extend our experiments for different processor types or different compilers for the same processor, our investigation was not intended to be an exhaustive survey of all possible compilers, cores, and SCHM architectures. Rather, we are motivating some basic findings that we believe will extend to many if not most other situations. It is intuitive, in light of these experimental results, that compilers play a key role in smoothing out variations in instruction mixes across a wide variety of behavior types, similar to the way architectural features common to all instructions that consume large amounts of power tend to smooth the variation in power consumption of individual instructions. Modeling at the program fragment level, we experience both of these effects for compiled fragments.

We had assumed multiple power signatures could be used and applied to separate program fragments, as in approach A2 of Section 4, in order to improve accuracy. Using two power signatures to group the benchmarks in Fig. 7 in this manner, for example, could decrease the average error in power estimation to only 3 percent per benchmark.

However, using an extra signature to model the power of a group of fragments that do not frequently execute will have little impact on the overall error. Since compilers tend to smooth instruction mixes generated from source-level code, the need for multiple classes of power signatures will most likely arise from some extraordinary condition, such as a fully hand-coded thread. Under these conditions, a designer is likely hand-tuning a thread with great knowledge of the underlying architecture. The thread is likely being performance-tuned or power-tuned (if not both). In this case, the average power consumption will not likely hold, but the reason it does not hold is because some unique situation is being exploited. Presumably, the reason it is being exploited is because the situation arises frequently enough for it to be worth the custom design time to bypass the compiler. When applications are tuned to architectures in this way, designers can assign signatures to fragments as a part of the design process, based upon knowledge of the unusual instruction mix. One important example of this is when a processor is in an idle state. In this case, designers may develop threads that execute *nops* or other instructions that limit memory access and so are designed to consume less power while the processor is waiting for something to happen, but not turned off. We use this type of signature in our example of Section 7. For other extraordinary situations, even when processors or fragments do not yet exist, designers may predict the effects of a code fragment executing on a processor resource so long as unique relationships can be drawn between physical systems and high-level models. For example, the use of a custom superinstruction such as a *DCT* (Discrete Cosine Transform) instruction might warrant a different power signature when that instruction is used frequently. This is analogous to the use of approach A3 from Section 4. At the system-level, this involves the identification of "super patterns" of instructions, which is

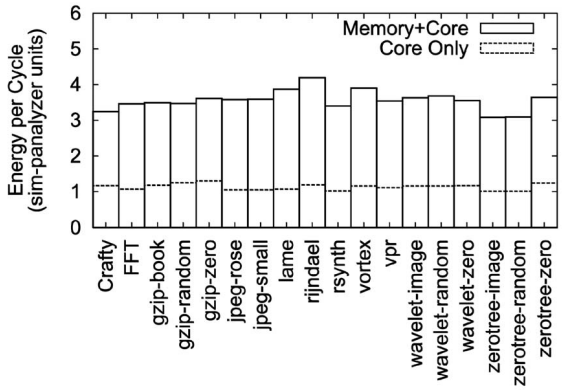


Fig. 8. Benchmark core and memory energy/cycle cost.

clearly more difficult, in general, than the identification of traditional architectural classes of instructions such as *Branch, Multiply*, or *Add*. However, as with the previous example of software that is not compiler-generated, these superinstructions are likely known at design time and, therefore, will be built into high-level models instead of automatically recognized.

We also theorized that memory access patterns would prove to be a differentiator of thread types. Fig. 8 compares the total energy per cycle cost of the core and memory subsystem together to the energy per cycle cost of the core alone. When the power used by the memory subsystem is considered, the variance in energy per cycle cost is on the scale of 1.11 units, compared to 0.24 units when only the core power dissipation is considered; the cost of the memory subsystem dominates, suggesting that fragments could be grouped by their expected behavior given the characteristics of a memory subsystem, ignoring the power consumption behavior of the core entirely. This generalizes to any subsystem with which a fragment may interact in some way.

Since interinstruction interaction at the microarchitectural level is analogous to interfragment interaction at the system level, we examined the effects of interinstruction interaction on our results. We have observed in our experiments that only one-third of the variation in the core energy in Fig. 7 is attributable to differences in the counts of instructions executed. It follows that the remaining two-thirds must be due to interinstruction and other instruction-architecture dependencies, e.g., the frequency of pipeline stalls. While, in this case, the overall impact of the variation is quite small, again leading to the conclusion that the same mix of interinstruction patterns appear across all compiled benchmarks, it indicates that instruction dependencies are a significant source of error in power modeling. At the system level, the cache state may be the most likely system-level analogy to capture interfragment effects. If, as was the case when considering a core and memory system, the power consumption of cache systems dominates that of cores, improved modeling of interfragment effects may reduce to improved modeling of caches at high levels. The application of knowledge of interfragment interaction, which is analogous to the application of context to inform power modeling at the system level (A4 of Section 4), may ultimately prove to be the most important factor in achieving acceptable high-level modeling. In any case, our

results show that it may be far more important to model the system-level effects of interfragment interaction than the instruction-level power consumption on individual cores.

5.4 Summary of Power Annotation Experimentation

Many problems remain to be solved in accurate high-level performance modeling. Our objective was not to solve all of the problems, but to discover those approaches that would likely have the most impact so that we could begin with a high-level modeling basis and work toward the next steps. Interestingly, from both our experimental results and as leakage current comes to dominate, the key to both power modeling and power management at this level of design may be the ability to characterize how long a processor is on or off, rather than what it is processing. The only exception may be when a processor is in some sort of sleep or idle state, in which case, it is processing a pathological set of instructions or reduced to even less activity, or when custom hardware is included well above the traditional instruction level. In both cases, power annotation may be a by-product of the design customization instead of a number automatically determined from source code. However, and as with performance tradeoffs of the use of multiple, cooperating PEs in lieu of single PEs, the cost of the cooperation will be a key factor to model in these designs. Thus, high-level power modeling may best focus efforts on modeling memory access patterns for a variety of parallel processing architectures, instead of modeling the PEs at the ISS-level.

6 Energy Modeling in MESH

Based upon the results of the previous section, we have added extensions to MESH that estimate energy consumption, resulting in a first-generation high-level power-performance simulator. While there is more work to be done, we show in Section 7 how MESH's foundation enables designers to explore SCHMs in new ways by using a new design strategy.

Energy is calculated in MESH as the power signature associated with a given fragment, multiplied by the time that fragment is executed over the duration of simulation, i.e., the performance of the fragment. All processors are modeled with at least one power signature. Execution time is determined by our high-level performance model, MESH. While the average power dissipation (signature) of a given fragment is assumed to be constant over the fragment, the execution time of all fragments can be data-dependent, affecting the execution time and, thus, the power dissipation calculated by MESH. Total system power is then summed over all PEs.

In the case of the example described in Section 7, we found two signatures were sufficient for each processor in the system (this is discussed in Section 7.2). One represents all software compiled for a given processor, while the other represents an idle, but still "on" condition.

Current ISS-level tools such as Sim-Panalyzer include the overhead associated with accessing memory external to a chip. When considering multiple processor cores on a single chip, the memory overhead would be far less. The cost of accessing memory (in terms of both time and power) will likely be the single largest differentiating factor in a designer's decision to use multiple, coordinated cores in lieu of one, which is the central question in low-power system-level design. In MESH, we currently include the ability to model time penalties associated with memory contention [10]. Since our power calibration currently includes off-chip memory access penalty, our memory access overhead is worse than for PE cores on a single chip. In order to provide a basis for evaluating a wider class of system-level design strategies than are included in this paper, the thread-level modeling of memory access power consumption and other interprocessor communications patterns will need to be addressed. And yet, one of the contributions of this paper is to motivate that reference models (or reference designs) for the calibration and verification of high-level power modeling of on-chip interprocessor coordination and interactions may be more important than instruction-accurate processor power models.

In the example in the next section, we model the impact of memory accesses as if they were off-chip, under the somewhat idealized workload of largely independent jobs (as part of an overall document). Our objective at this early stage is to make the case for simulation that supports the development and application of higher level approaches to optimization of performance and power on single chip designs by pointing in the direction of gains that can be achieved there that could not be exploited otherwise.

7 Example Design Strategy: Spatial Voltage Scaling

The real value of MESH is more than simply showing how one design point can be achieved faster than at the ISS-level or below, but in permitting designers to think in entirely new ways. New levels of modeling and simulation open up possibilities for new design strategies because of the ability to manipulate design elements at drastically reduced levels of detail. We introduce one such strategy, *spatial voltage scaling*, around which we orient a set of experiments that show how the design space described in Section 2 might be narrowed for one example system. Dynamic voltage scaling is a technique that matches the execution frequency (and, so, voltage level) of a given processor to the demands of the application at runtime [26]. When performance is less critical, the processor voltage is scaled down so that it takes longer for a portion of the application to execute. The result is a finite set of voltage levels and a dynamic decision-making strategy for when the different voltage levels available in the processor are applied.

Spatial voltage scaling is inspired by a blend of dynamic voltage scaling and processor-rich design. We describe processor-rich design as the well-known hypothesis that n processors executing at a frequency f/n (and, thus, a lower voltage) can be substituted for a single processor executing at frequency f for nearly equivalent overall performance and greatly reduced overall power. Of course, this is the case only if the application lends itself to parallel execution and if the cost of the parallelization does not exceed the benefits. However, this is the reason for needing a simulation environment that can permit designers to effectively explore this space. In spatial voltage scaling, instead of dynamically scheduling different voltage levels on a single processor, multiple processors operate at a variety of statically determined voltage levels that match the anticipated demands of the system under a variety of

testbench scenarios; operating voltages are spatially distributed instead of multiplexed in time.

Spatial voltage scaling starts with the observation that many systems process sets of homogeneous job types. However, data-dependent execution times, variation in the job sizes, and variation in the arrival rates of jobs can lead to system power inefficiencies if a homogeneous set of processor resources is utilized. Thus, instead of substituting n processors executing at f/n in place of a single processor executing at f, spatial voltage scaling seeks to find not only n, but a set of f_i, one for each processor in the system. Thus, n processors may be substituted for a single processor where the set of processors has a set of frequencies, $\{f_1, \ldots, f_n\}$. The frequency of each processor in the system is fixed at design time, potentially resulting in a heterogeneous set of frequencies on homogeneous processor types. In reality, if the frequency reduction is drastic, a processor would likely be replaced by a simpler processor consisting of a less complex architecture.

7.1 Experiments

We ran a set of experiments to test the hypothesis that a spatially voltage scaled design could reduce overall power consumption while maintaining performance. While it is intuitive that this would be the case for some designs, the ability to efficiently explore this hypothesis for a given design has been limited without the high-level power-performance simulator, MESH.

Our experimental system is part of a document management system designed to compress documents that consist of a variable mixture of text and image elements. A document is not considered processed until all of its constituent parts are compressed. We begin with a design space related to Fig. 1 as:

$$
\begin{aligned}
A &= \{gzip, wavelet\ transform, zerotree\ quantization\},\\
T &= \{T_1, T_2, T_3\},\\
S &= \{\text{job-type-aware } (\mathrm{S_{JT}}) \text{ [with dynamic shutdown } (\mathrm{S_{JTDS}})],\\
&\qquad \text{job-size-aware } (\mathrm{S_{JS}}) \text{ [with dynamic shutdown } (\mathrm{S_{JSDS}})]\},\\
R &= \{\{\text{number of, type of, and operating frequency of}\\
&\qquad \text{processors}\}, \text{shared memory communication}\},
\end{aligned}
$$

where there is a rich set of design trade-offs in the hardware, software, and scheduling decisions of the system. Document arrival times, job sizes, and data content are all variable as well. We create a baseline design around which we focus on validating our spatial voltage scaling hypothesis, which, in turn, demonstrates the value of our high-level simulator.

The baseline architecture consists of one ARM and one DSP, each executing at 233 MHz with a 1.65 V supply voltage, consuming 4.49 W and 4.62 W for the ARM and DSP, respectively. The performance of the DSP is modeled based on a Renesas M32R that contains various DSP class instructions such as a MAC; the power consumption of the DSP is modeled after that of the ARM, but adjusted to account for the presence of DSP class instructions. Throughout the paper, we simply refer to this type of PE as a DSP.

The baseline software design consists of three thread types that can execute on either processor type. The text portions are processed by a thread that runs *gzip* compression. Each image compression requires two threads: *wavelet* transform and *zerotree* quantization. The execution time of a given *gzip* job is approximately twice as fast on an ARM as it is on a DSP. Conversely, the execution time of the *wavelet* is approximately twice as fast on a DSP as on an ARM. An ARM and a DSP are equally adept at *quantization*. An SCHM consisting of an ARM processor and a DSP has been previously shown to provide performance improvement when jobs are sometimes scheduled on nonideal processors [10]. The signatures of each thread on a given processor did not vary enough to warrant the use of more than two signatures on a single processor.

Each document is injected into the system at a Poisson rate defined by the arrival rate in T. T_1 represents a class of documents where job size is evenly distributed; it contains three text-image pairs: one large, one medium, one small. T_2 represents a class of documents where the amount of data in each size category is evenly distributed; it contains seven text-image pairs: one large, two medium, four small. T_3, which contains 15 large images, represents a worst-case document. T_3 is worst case because each processed image requires two jobs and each job is large. These testbenches were chosen to expose the weaknesses of particular classes of architecture configurations.

S_{JT} and S_{JS} are designed to reduce overall document latency by making intelligent job-to-processor mapping decisions. S_{JT} attempts only to map jobs to optimal processors based on job type. S_{JS} extends S_{JT} by taking into account the size of the job when making mapping decisions. Each PE is assigned an ideal range of job sizes (job size bin) proportional to PE operating frequency and expected job sizes. If S_{JS} cannot find a processor of optimal type and job size bin, it behaves exactly like S_{JT}. Both S_{JT} and S_{JS} are extended with dynamic shutdown [33]. S_{JTDS} and S_{JSDS} are designed to dramatically reduce energy consumption by turning off whole PEs if they have been idle longer than a threshold. A typical cost associated with restarting processors that have been turned off is on the order of milliseconds; we conservatively use a value of 20 ms in our experiments.

The majority of our design exploration takes place in R. All configurations, R_k, had the same number of ARM as DSP processors, organized in pairs according to operating frequency. The devices complement each other in computational ability with respect to the application, naturally leading to pairing. The architecture configurations we tested can be broken into four classes: base, static voltage scaling, spatial voltage scaling, and static/spatial hybrid configurations. Base configurations consisted of one to three pairs of processors operating at 233 MHz. Statically scaled configurations consisted of three to five pairs of processors operating at some $f < 233$ MHz. Base and statically scaled configurations use S_{JT} (or S_{JTDS}) for job scheduling. Spatially scaled configurations consisted of three pairs of processors operating at some $f < 233$ MHz, $2f/3$, and $f/3$. The final class of hybrid configurations consisted of four pairs of processors, two of which operated at some $f < 233$ MHz and two of which operated at $2f/3$ and $f/3$, respectively. Note that spatial scaling as defined above need not be restricted in this fashion; spatial voltage scaling is a design heuristic, as is our specific application of it. Spatially scaled and hybrid configurations use S_{JS} (or S_{JSDS}) for job scheduling.

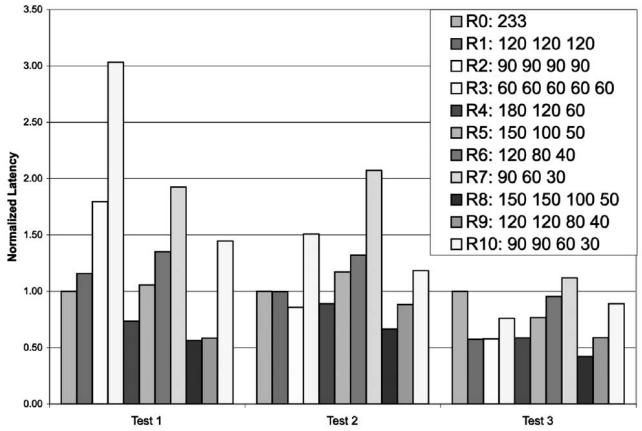


Fig. 9. Latency across all testbenches.

7.2 Power Model

Using the results of our investigation, we used two power signatures to model a processor that was considered "on." Processors were modeled to consume no energy when they were off. The first power signature characterizes the cost of executing the different programs in the example application. This signature was calculated by taking the average energy per cycle cost of the programs. This is reasonable since these programs differ by no more than 3 percent in their power consumption and were all compiled with *gcc*. A second power signature was used to model the power consumption of processors that are not turned off, but are not computing anything useful, i.e., they are idle, waiting for something to happen, such as a job to appear in the queue. This signature captures processor behavior when portions of the processor go relatively unutilized, but the processor is not asleep and, therefore, dissipates power. In our model, the idle job checked the job queue periodically, but otherwise executed instructions that did not require external memory accesses. Thus, the processor had a different power signature during idle behavior. The power dissipation of this signature is determined using Sim-Panalyzer and is a measure of the energy dissipated per cycle when the processor is executing an instruction mix uniquely characteristic of when it is idle. This mix is represented in our example as the continuous execution of *nops*.

In our experiments, we have two PE types: an ARM and a DSP. Sim-Panalyzer is limited to modeling power in an ARM. Because of this restriction, we first determined the power consumption of an ARM using Sim-Panalyzer and assumed that the power behavior of the DSP was the same as that of the ARM, except for the power consumption of DSP class instructions such as a multiply-accumulate (*MAC*). To determine the impact of this instruction, we estimated the cost of a *MAC* and measured with a Reneasas M32R ISS how often the *MAC* is executed and adjusted the power signature of the DSP accordingly.

7.3 Model Parameters

Determining valid processor operating points is essential for calculating processor power dissipation. In our experiments, we chose the minimum supply voltage possible for a given clock frequency. In modern technologies, maximum clock frequency varies approximately linearly with supply voltage [32]. Using this, we determined the correct frequency to voltage relationship using data available for two modern processors capable of dynamic voltage scaling [33], [34]. This relationship is shown in Fig. 10.

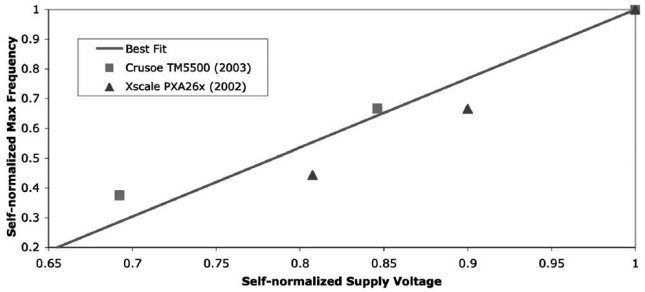


Fig. 10. Linear fit of frequency to voltage relationship.

Since the voltage and frequency ranges for the two processors are different, the data for each processor was self-normalized. Each processor therefore has a data point at (voltage, frequency) = (1, 1). To make our linear approximation valid across the entire range of possible frequency and voltage values, the line is constrained to pass through the point (1, 1). These constraints yield the linear approximation $f_{rel} = 2.32v_{rel} - 1.32$, where f_{rel} and v_{rel} represent fractions of the maximum frequency f_{max} and supply voltage v_{max}. For example, if $f_{max} = 100$ and $f = 50$, then $f_{rel} = 0.5$. With this equation, we can determine the operating voltage v for any f, given f_{max}, v_{max}. With v and maximum power dissipation P_{max} (determined as outlined in Section 7.2), it is possible to determine the power dissipation P for a processor operating at f.

Once f is chosen, we find v by first substituting f/f_{max} and v/v_{max} for f_{rel} and v_{rel}, respectively, then solving for v.

$$v = \left(1.32 + \frac{f}{f_{max}}\right)\frac{v_{max}}{2.32}.$$

Then, we substitute into $P = Cfv^2$, yielding

$$P = Cf\left(\left(1.32 + \frac{f}{f_{max}}\right)\frac{v_{max}}{2.32}\right)^2,$$

where C is determined a priori to be

$$C = \frac{P_{max}}{\left(f_{max}v_{max}\right)^2}.$$

In this fashion P can be found for a processor operating at an arbitrary f, given we know f_{max}, v_{max}, and P_{max}.

7.4 Results

We ran all three testbenches on 10 architectures and a baseline architecture. The subset of architectures with the best latency in these first tests was then chosen for a second round of tests. In the second set of tests, the dynamic shutdown scheduling policies, S_{JTDS} and S_{JSDS}, were applied to the subset of configurations found in the first set of tests and a set of enhanced base cases. The architecture deemed optimal at this stage using a combination of normalized latency, energy, and latency-energy product measurements was judged the overall optimal design.

In the charts that follow, the legend shows the architecture parameters R_k and the scheduling parameter S_j, when relevant. Each number in the legend represents a pair of

TABLE 1
Architecture Classes

Base Cases	R_0 R_{11}
Static Scaling	R_1 R_2 R_3
Spatial Voltage Scaling	R_4 R_5 R_6
Hybrid Static/Spatial	R_8 R_9

ARM and DSP PEs. For example, for the base case R_0, "233" denotes two PEs: one ARM and one DSP, each operating at 233 MHz. Table 1 shows the groupings of architectures by design strategy.

We begin by eliminating all designs where normalized average latency, as shown in Fig. 9, is worse than the base case's for any of the three testbenches. The only architectures that perform as well as or better than the base case across all T_n are R_4, R_8, and R_9. The entire statically scaled class of architectures (R_1-R_3) was eliminated because of its performance in T_1.

Once the set of architectures is narrowed, we introduce dynamic shutdown so individual PEs are turned off and consume no power after a period of inactivity. As shown in Fig. 11, we add two new architectures (R_{11} and R_{12}) that consist of four and six PEs operating at 233 MHz using S_{JTDS} and retain the original base case, R_0, that does not use dynamic shutdown. We ran these new configurations on all three testbenches and show the results from T_1 in Fig. 11. While R_0, R_{11}, and R_{12} using S_{JTDS} have good latency, there is a significant energy penalty for having so many high-power PEs. The three architectures that met the latency requirement in Fig. 9 (R_4, R_8, and R_9) clearly also outperform all architectures using 233 MHz PEs (R_0, R_{11}, and R_{12}) in terms of latency-energy product when S_{JTDS} and S_{JSDS} are applied, respectively.

While we do not show the results for T_2 using dynamic shutdown due to space limitations, the general trends were the same as in Fig. 11. The results from T_1 and T_2 are so similar when dynamic shutdown is applied that we only consider results for T_1 and T_3 when choosing the final optimal solution.

Fig. 12 shows the energy and latency-energy product for our final candidate architectures exercised by T_1 and T_3. R_9 consumes only 45 percent as much energy as the base case under T_3 and 27 percent under T_1. Its latency-energy product is 26 percent of the base case's for T_3—a 49 percent improvement over the next best performer. R_9 using S_{JSDS}

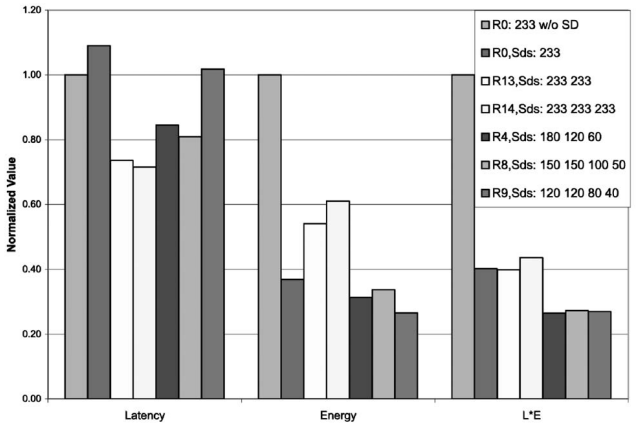


Fig. 11. Dynamic shutdown results for T_1.

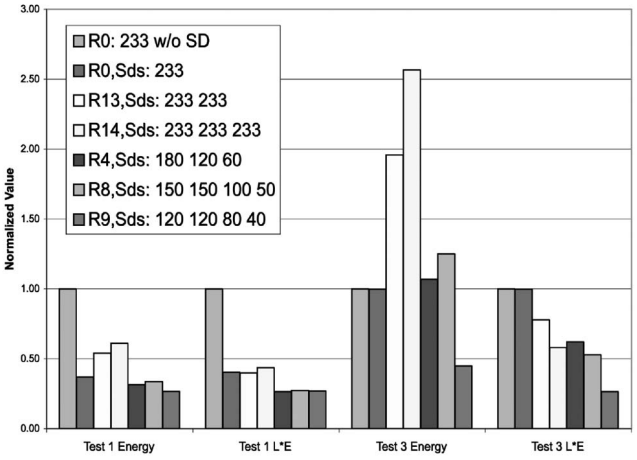


Fig. 12. Energy and latency-energy for T_1 and T_3.

is the overall optimal solution, performing as well as R_4 and R_8 under T_1, while excelling in comparison to them under T_3.

In summary, our design tool and spatial voltage scaling design strategy enabled us to converge on an optimal system design that achieves both an average 15 percent latency improvement and an average 66 percent power improvement over the baseline design of a single ARM/DSP pair executing at 233 MHz, even when each design uses a dynamic shutdown scheduling policy. This optimal design is a hybrid statically/spatially scaled system consisting of four ARM/DSP pairs capable of dynamic shutdown: two pairs operating at 120 MHz, one pair at 80 MHz, and one pair at 40 MHz. It would have been highly unlikely to discover, via intuition, analytical models or at the ISS-level.

8 Conclusion

We formed the basis for a set of experiments that enabled us to reach some conclusions about the level of detail required to achieve thread-level power-performance simulation of SCHMs and where researchers should focus the development of detailed reference models that can be used to calibrate high-level models. Contrary to our original hypothesis that we would need to classify and annotate program fragments by type, we found that it can be sufficient to use only a single number that characterizes the power consumption of a given compiled code fragment on a given processor in a system. Modeling power in this fashion results in an average per benchmark error of 6 percent, with a maximum error of 15 percent, largely due to the role compilers play in smoothing out instruction mixes and because microarchitectures tend to consume a high percentage of power in fetch and decode. The use of additional power annotations, or power signatures, for different fragments can arise when code is developed by designers by hand, such as when a processor is executing a custom, low-power, idle thread. However, in this case, power signatures can be obtained at the time the custom fragment is developed. Thus, the dominant factor required to calculate energy consumption in a multiprocessor is the performance of the thread executing on that PE, i.e., the time it spends executing.

The challenge does not lie in modeling the power consumptions of individual instructions or instruction sequences on a processor model since power can be

obtained by measuring average values over representative benchmarks. The challenge lies in developing reference models for calibration and modeling, at the thread level, of the power and performance of the coordination and interactions of collections of PEs, whether via a memory hierarchy or an on-chip network.

We also introduced a novel design strategy, spatial voltage scaling, enabled by the early, high-level power-performance simulator, MESH. The power modeling extensions to MESH as described in this paper enabled the application of this design strategy. The final design was discovered using spatial voltage scaling and dynamic shutdown. It reduces both power and latency over a baseline design that used dynamic shutdown, but did not use spatial voltage scaling. Our optimal design achieves an average of 66 percent energy improvement over a baseline case and an average of 15 percent latency improvement. The latency-energy product of our final design is a 49 percent improvement over the next best performer. The optimal design is a hybrid statically/spatially scaled system. Its discovery required a combination of a strategic way of exploring the design space, enabled by high-level simulation. Analytical models would not have captured the scheduling decisions, which were an important part of the design space, and ISS-level simulation would have been prohibitively detailed to develop and slow to execute, even for this relatively simple design.

ACKNOWLEDGMENTS

This work was supported in part by ST Microelectronics, General Motors, and the US National Science Foundation (NSF) under Grants 0103706 and CNS-0406384. 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. The authors would like to thank the other members of the MESH team, Donald E. Thomas and Alex Bobrek, in particular, for their careful reading of this paper. The authors would also like to thank the reviewers for their suggestions. They made the paper better.

REFERENCES

[1] R. Bergamaschi, I. Bolsens, R. Gupta, R. Harr, A. Jerraya, K. Keutzer, K. Olukotun, and K. Vissers, "Are Single-Chip Multiprocessors in Reach?" *IEEE Design and Test of Computers,* vol. 18, no. 1, pp. 82-89, Jan./Feb. 2001.

[2] W. Wolf, "How Many System Architectures?" *Computer,* vol. 36, no. 3, pp. 93-95, Mar. 2003.

[3] F. Karim, A. Mellan, A. Nguyen, U. Aydonat, and T. Abdelrahman, "A Multilevel Computing Architecture for Embedded Multimedia Applications," *IEEE Micro,* vol. 24, no. 3, pp. 56-66, May-June 2004.

[4] J.M. Paul, "Programmer's View of SoCs," *Proc. Int'l Conf. Hardware/Software Codesign and System Synthesis (CODES-ISSS),* pp. 159-161, Oct. 2003.

[5] J.L. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *Computer,* vol. 33, no. 7, pp. 28-35, July 2000.

[6] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proc. 2001 IEEE Int'l Workshop Workload Characterization (WWC-4),* pp. 3-14, Dec. 2001.

[7] J.M. Paul, D.E. Thomas, and A. Bobrek, "Benchmark-Based Design Strategies for Single Chip Heterogeneous Multiprocessors," *Proc. Second IEEE/ACM/IFIP Int'l Conf. Hardware/Software Codesign and System Synthesis, 2004,* pp. 54-59, 2004.

[8] A.S. Cassidy, J.M. Paul, and D.E. Thomas, "Layered, Multi-Threaded, High-Level Performance Design," *Proc. Design, Automation and Test in Europe Conf. and Exhibition, 2003,* pp. 954-959, 2003.

[9] J.M. Paul, A. Bobrek, J.E. Nelson, J.J. Pieper, and D.E. Thomas, "Schedulers as Model-Based Design Elements in Programmable Heterogeneous Multiprocessors," *Proc. Design Automation Conf., 2003,* pp. 408-411, June 2003.

[10] A. Bobrek, J.J. Pieper, J.E. Nelson, J.M. Paul, and D.E. Thomas, "Modeling Shared Resource Contention Using a Hybrid Simulation/Analytical Approach," *Proc. Design, Automation and Test in Europe Conf. and Exhibition, 2004,* vol. 2, pp. 1144-1149, Feb. 2004.

[11] J.M. Paul, S.N. Peffers, and D.E. Thomas, "A Codesign Virtual Machine for Hierarchical, Balanced Hardware/Software System Modeling," *Proc. Design Automation Conf.,* pp. 390-395, 2000.

[12] C.L. Seitz, "System Timing," *Introduction to VLSI Systems,* C. Mead and L. Conway, eds., Reading, Mass.: Addison-Wesley, 1980.

[13] C. Brandolese, W. Fomacian, F. Salice, and D. Sciuto, "An Instruction-Level Functionality-Based Energy Estimation Model for 32-Bits Microprocessors," *Proc. Design Automation Conf.,* pp. 346-350, 2000.

[14] X. Liu and M.C. Papaefthymiou, "A Static Power Estimation Methodology for IP-Based Design," *Proc. Design, Automation and Test in Europe, 2001, Conf. and Exhibition,* pp. 280-287, 2001.

[15] E. Macii, M. Pedram, and F. Somenzi, "High-Level Power Modeling, Estimation, and Optimization," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 17, no. 11, pp. 1061-1079, Nov. 1998.

[16] I. Kadayif, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, and A. Sivasubramaniam, "EAC: A Compiler Framework for High-Level Energy Estimation and Optimization," *Proc. Design, Automation and Test in Europe Conf. and Exhibition, 2002,* pp. 436-442, 2002.

[17] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, "Cosimulation-Based Power Estimation for System-on-Chip Design," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 10, no. 3, pp. 253-266, June 2002.

[18] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. 27th Int'l Symp. Computer Architecture, 2000,* pp. 83-94, 2000.

[19] W. Ye, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin, "The Design and Use of Simplepower: A Cycle-Accurate Energy Estimation Tool," *Proc. Design Automation Conf., 2000,* pp. 340-345, 2000.

[20] N. Kim, T. Kgil, V. Bertacco, T. Austin, and T. Mudge, "Microarchitectural Power Modeling Techniques for Deep Sub-Micron Microprocessors," *Proc. In'tl Symp. Low Power Electronics and Design (ISLPED),* pp. 212-217, Aug. 2004.

[21] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0," *SIGARCH Computer Architecture News,* vol. 25, no. 3, pp. 13-25, 1997.

[22] J.T. Russell and M.F. Jacome, "Software Power Estimation and Optimization for High Performance, 32-Bit Embedded Processors," *Proc. Int'l Conf. Computer Design: VLSI in Computers and Processors (ICCD '98),* pp. 328-333, Oct. 1998.

[23] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon, "Energy Estimation and Optimization of Embedded VLIW Processors Based on Instruction Clustering," *Proc. Design Automation Conf.,* pp. 886-891, 2002.

[24] G. Qu, N. Kawabe, K. Usami, and M. Potkonjak, "Function-Level Power Estimation Methodology for Microprocessors," *Proc. Design Automation Conf.,* pp. 810-813, 2000.

[25] M.T.-C. Lee, V. Tiwari, S. Malik, and M. Fujita, "Power Analysis and Minimization Techniques for Embedded DSP Software," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 5, no. 1, pp. 123-135, Mar. 1997.

[26] T.D. Burd, T.A. Pering, A.J. Stratakos, and R.W. Brodersen, "A Dynamic Voltage Scaled Microprocessor System," *IEEE J. Solid-State Circuits,* vol. 35, no. 11, pp. 1571-1580, Nov. 2000.

[27] V. Tiwari and M.T.-C. Lee, "Power Analysis of a 32-Bit Embedded Microcontroller," *Proc. Design Automation Conf., 1995, Proc. ASP-DAC '95/CHDL '95/VLSI '95, IFIP Int'l Conf. Hardware Description Languages; IFIP Int'l Conf. Very Large Scale Integration, Asian and South Pacific,* pp. 141-148, 1995.

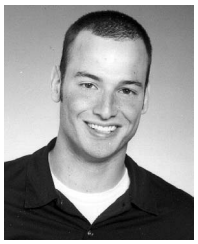
[28] V. Tiwari, S. Malik, A. Wolfe, and M.T.-C. Lee, "Instruction Level Power Analysis and Optimization of Software," *Proc. Ninth Int'l Conf. VLSI Design,* pp. 326-328, Jan. 1996.

[29] J. Flinn and M. Satyanarayanan, "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications," *Proc. Second IEEE Workshop Mobile Computing Systems and Applications (WMCSA '99),* pp. 2-10, Feb. 1999.
[30] A.J. KleinOsowski and D.J. Lilja, "MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research," *Computer Architecture Letters,* vol. 1, June 2002.
[31] M.B. Srivastava, A.P. Chandrakasan, and R.W. Brodersen, "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 4, no. 1, pp. 42-55, Mar. 1996.
[32] A. Waizman and C. Chee-Yee, "Package Capacitors Impact on Microprocessor Maximum Operating Frequency," *Proc. 51st Electronic Components and Technology Conf.,* pp. 118-122, 2001.
[33] Intel PXA26x Processor Design Guide, ftp://download.intel.com/design/pca/applicationsprocessors/manuals/27863902.pdf, Oct. 2002.
[34] Transmeta Crusoe Processor Model TM5500 Processor Product Brief, http://www.transmeta.com/pdfs/TM5500_productbrief_030206.pdf, Feb. 2003.
[35] T. Weiyu, R. Gupta, and A. Nicolau, "Power Savings in Embedded Processors through Decode Filter Cache," *Proc. Design, Automation and Test in Europe Conf. and Exhibition,* pp. 443-448, 2002.

Brett H. Meyer received the BS degree in electrical engineering from the University of Wisconsin-Madison in 2003. He is currently a graduate student in the Electrical and Computer Engineering Department at Carnegie Mellon University, working on energy efficient designs of single chip heterogeneous multiprocessor systems. He is a student member of the IEEE.

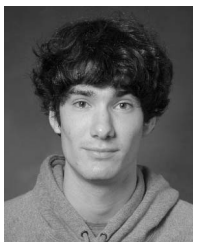
Joshua J. Pieper received the BS degree in computer engineering from the University of Missouri-Rolla in 2002 and the MS degree in electrical and computer engineering from Carnegie Mellon University in 2004.

JoAnn M. Paul received the BS degree in electrical engineering from the University of Pittsburgh (summa cum laude) in 1983, the MS degree in electrical and computer engineering from Carnegie Mellon University in 1988, and the PhD degree in electrical engineering from the University of Pittsburgh in 1994. She has been a member of the Research Faculty at Carnegie Mellon University since 2000. She has prior industrial experience in the development of hardware and software for a variety of computer systems and, since entering academia, has published more than a dozen papers. Her research interests are in high-level modeling, simulation, and design of single chip heterogeneous multiprocessors. She is a licensed Professional Engineer in the Commonwealth of Pennsylvania, a member of the IEEE and the IEEE Computer Society.



Jeffrey E. Nelson received the BS degree in electrical and computer engineering from Rutgers University, New Brunswick, New Jersey, in 2002 and the MS degree in electrical and computer engineering from Carnegie Mellon University (CMU) in 2003. He is currently a graduate student in the Electrical and Computer Engineering Department at CMU with research interests in modeling, test generation, design for manufacturing, and yield learning. He is a student member of the IEEE.

Sean M. Pieper received the BS and MS degrees, both in electrical and computer engineering, from Carnegie Mellon University in 2003 and 2004, respectively. He is currently a graduate student in electrical and computer engineering at the University of Wisconsin-Madison with research interests in low power architectures for embedded applications.



Anthony G. Rowe received the BS degree in electrical and computer engineering from Carnegie Mellon University (CMU) in 2003. He is currently a graduate student in the Electrical and Computer Engineering Department at CMU, working on embedded real-time systems.