Design and validation of a performance and power simulator for PowerPC systems

H. Shafi
P. J. Bohrer
J. Phelan
C. A. Rusu
J. L. Peterson

This paper describes the design and validation of a performance and power simulator that is part of the Mambo simulation environment for PowerPC® systems. One of the most notable features of the simulator, designated as ***Tempo****, is the incorporation of an event-driven power model. Tempo satisfies an important need for fast and accurate performance and power simulation tools at the system level. The power and performance predictions from the simulated model of a PowerPC 405GP (or simply 405GP) were validated against a 405GP-based evaluation board instrumented for power measurements using 42 application/dataset combinations from the EEMBC benchmark suite. The average performance and energy-prediction errors were 0.6% and −4.1%, respectively. In addition to describing Tempo, we show examples of how well it can predict the runtime power consumption of a 405GP microprocessor during application execution.*

1. Introduction

Computer architects and software developers are faced with a dilemma. In addition to performance, which is the primary design goal of high-end microprocessors, systems, and software, power has emerged as a second primary design metric, especially for embedded systems. Unfortunately, there has been a lack of practical simulation and profiling tools for "what-if" power analysis of new and existing architectures. Performance architectural simulators typically include cycle-accurate models of varying complexity for the systems under investigation. If these simulators are to be practical enough to enable the study of complete applications using realistic datasets, simulation speed will be critical (the classic tradeoff of detail vs. speed is often resolved in favor of the latter).

Further, the power simulation tools that exist today range from very detailed and slow transistor-level SPICE-like [1] models, to higher-level Verilog/VHDL circuit-level power simulators such as PowerTheater** [2] and PowerMill** [3], to the even higher levels of abstraction found in tools such as Wattch [4] that can be integrated with architectural simulators. Our goal has been to integrate cycle-accurate performance and power simulation at the minimum possible speed penalty. Our approach differs from that used in Wattch in that we further abstract the details of power simulation, because our target audience is architects and developers of operating systems and applications. The dilemma of the architects/researchers extends to developers of power-aware software on systems. While they are typically not as concerned about circuit details involved in arriving at power estimates, they want to know where power is dissipated in their applications. Very few tools can provide such feedback.

Event-based energy tracking models have previously been shown to provide good microprocessor power estimates [5]; hardware registers count the number of occurrences of a limited set of events, allowing an estimate of energy usage over time. Unfortunately, such models are not useful for studying new architectures or systems that do not include event-tracking support in hardware. Since many cycle-accurate simulators are event-

0018-8646/03/$5.00 © 2003 IBM

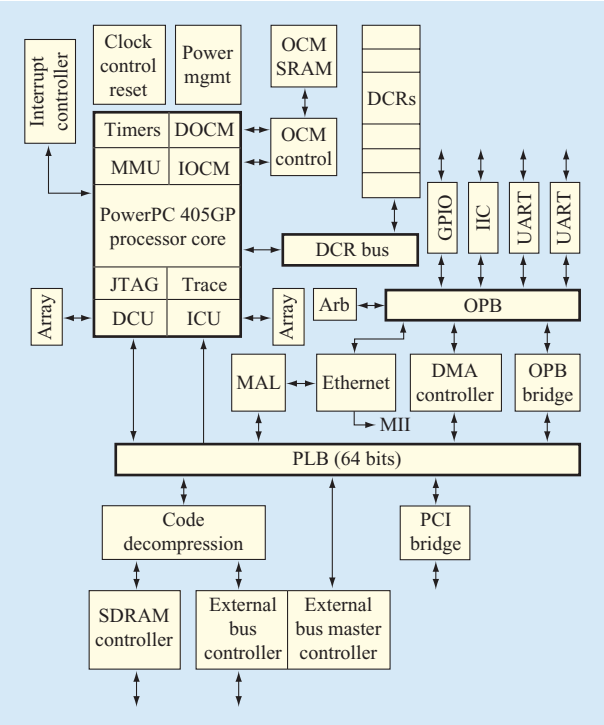

Figure 1

Block diagram of the PowerPC 405GP processor.

driven, we sought to investigate the possibility of modeling the power consumption of a microprocessor by associating energy costs with the occurrence of certain architectural events. If successful, this power modeling approach could be performed at almost no additional simulator runtime overhead; since the important architectural events are already modeled for timing estimates, energy estimates could be computed simply by some additional counting. Our methodology should be extendable to future microprocessors by relying on some of the other, more detailed power simulation techniques to generate our event-based power model during early design stages.

To determine the feasibility of our approach, we decided to model an existing microprocessor: the core processor found in the PowerPC* 405GP [6] system-on-a-chip. Since the 405GP is an existing processor, we could validate our simulator against actual hardware. Our colleagues at the IBM Austin Research Laboratory had designed the Pecan board, a 405GP-based system that has provisions for power measurements. Although the 405GP processor does not include hardware performance counters, it is a relatively simple in-order pipelined processor. If we could build a cycle-accurate version of the 405GP, our simulator could be used for the performance/power tuning of applications and operating systems, despite the lack of hardware performance counters. Enabling such evaluation methodology is very desirable for embedded systems such as the 405GP.

Our efforts have resulted in the development of *Tempo*, an execution and event-driven cycle-accurate simulator that is part of the Mambo PowerPC simulation environment of the IBM Austin Research Laboratory. Our validation results show that the processor core performance predictions of the simulator have an average error of 0.6%, with a standard deviation of 2.5% on 42 Embedded Microprocessor Benchmarking Consortium (EEMBC) [7] benchmarks when compared to the hardware platform being modeled. The average error in energy predictions was −4.1%, with a standard deviation of 5.1%. In addition, we could model the transient power behavior of applications, which is not possible with many energy models. Tempo can bridge this gap because it can compute the power consumption on a per-cycle basis by accumulating event energies on every cycle.

The rest of the paper is organized as follows. Section 2 describes the Mambo simulation environment, with special emphasis on the Tempo model. Section 3 presents our timing validation methodology and results. Section 4 describes how the power model was created and presents power validation results. Related work is discussed in Section 5. Finally, we present concluding remarks in Section 6.

2. Mambo simulation environment

Mambo is an IBM proprietary full-system simulation toolset for the PowerPC architecture. It shares some of its roots with the PowerPC extensions added to the SimOS [8] simulator and is written completely in the C programming language. It can be run on many platforms, but we commonly run it on both PowerPC and x86 processors, under either AIX* or Linux**.

Mambo supports the 64-bit PowerPC processor and the 32-bit 405GP processor. The system is designed for multiple configuration options. Various PowerPC extensions and attributes such as vector multimedia extensions (VMX) support, hypervisor, cache geometries, segment lookaside buffers (SLBs) and translation lookaside buffers (TLBs) can be configured. Models of future PowerPC architectures can be created by selecting from the various configuration options. Further, Mambo includes models for disks [9], various system and I/O buses, Ethernet controllers, and Universal Asynchronous Receiver/Transmitter (UART) devices. When these models are combined with the full-architecture processor models, full-system simulations with real operating systems and applications are possible. Mambo is in active use by multiple research and development efforts at IBM. The

rest of this section concentrates on aspects of Mambo that are relevant to this paper.

One of the processor models supported by Mambo is the PowerPC 405GP, a 32-bit system-on-a-chip PowerPC processor used in embedded applications. **Figure 1** shows a block diagram of the system. Mambo simulates not only the instructions executed by the processor core, but also its interactions with its surrounding devices.

The 405GP simulator includes two processor models: Simple and Tempo. The former is a basic functional model that does not try to model complex timing aspects of program execution. To simulate an "add" instruction, for example, it simply fetches the two operands from simulated registers or memory, computes the sum, and sets the simulated registers to the results defined by the PowerPC instruction set. The Simple model assumes that each instruction takes one cycle and that access to memory is immediate and takes no time.

The advantage of the Simple model is its speed. On the EEMBC benchmark suite, we can simulate all of its 2,225,591,852 instructions from power-on to the end of all benchmarks in 15.5 minutes at about 2.4 million instructions per second on a 1.2-GHz AMD Athlon** processor-based system running Red Hat** Linux. The disadvantage of Simple is that it provides no information about the actual number of cycles needed to execute a set of instructions: All it does is count instructions, one instruction per cycle. The number of cycles reported by Simple is not affected by the type of instruction, memory accesses, caches, or any of the other features of the 405GP. Timers, I/O, and other external devices are correctly modeled with respect to time, but the timing aspects of the processor are not modeled.

The PowerPC 405GP can also be simulated by the Tempo model, which adds accurate timing information to the Simple model. The 405GP processor core implements the PowerPC instruction set using an in-order five-stage pipeline, as shown in **Figure 2**. Most instructions execute in one cycle, but some instructions (such as multiplication and division) require more cycles (e.g., division requires 35 cycles) and some functional units are pipelined. Although the 405GP core is a single-instruction in-order issue processor, it allows the overlap of load misses with independent instructions.

The Tempo model of this system includes cycle-accurate details of the processor core, including its pipeline, instruction fetch unit, branch predictor, instruction and data caches, memory management unit including both instruction-side and data-side TLBs, functional units, and timers. Further, the system memory, PLB bus, interrupt controller, memory controller, and memory subsystem are all modeled.

Correctly simulating the PowerPC 405GP pipeline and the complexities of overlapped execution requires more work on the part of Tempo. For the same EEMBC benchmark suite and computational platform mentioned above for Simple, Tempo requires 63.1 minutes, running at an average of 587 thousand instructions per second, a factor of 4× slower than Simple. In exchange for this longer execution time, it provides a more accurate simulation time of 3,579,669,994 cycles.

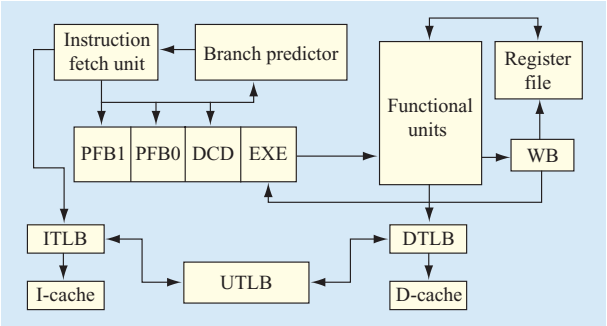

Figure 2

Simplified representation of the PowerPC 405GP pipeline.

Table 1 PowerPC 405GP settings for modeling and validation.

Parameter	*Value*
CPU core frequency	200 MHz
Processor local bus (PLB) frequency	66 MHz
Core voltage (used for measurements)	2.5 V
SDRAM speed/voltage	10 ns/3.3 V

3. Timing validation

Table 1 shows the PowerPC 405GP settings used for timing and power validation measurements during our experiments.

Tempo PowerPC 405GP model

The 405GP core is a relatively simple processor to simulate owing to its in-order issue architecture; however, discovering the timing details needed for a cycle-accurate model was a nontrivial task, primarily because of our inability to locate detailed architectural (behavioral) specifications. The timing information used for the processor model was gathered either from existing publicly available user manuals or through experimentation.

Whenever we could not find good documentation, we resorted to running carefully designed microbenchmarks that were used to identify the behavior of a specific aspect of the microarchitecture. A microbenchmark is designed

to allow the time associated with a specific event to be determined. For example, to measure the time for a cache miss, we can use a microbenchmark such as the following:

```
loop:
  lwz   r2, 0 (r3)
  addi  r2, r2, 1
  lwz   r2, 0 (r4)
  addi  r2, r2, 1
  lwz   r2, 0 (r5)
  addi  r2, r2, 1
  bctr  loop
```

This loop consists of three load instructions followed by instructions that use the loaded value. The load followed by a use causes the `addi` instruction to wait for the result of the load. The load instructions load from addresses held in registers `r3`, `r4`, and `r5`. To measure the latency of a load miss, we run this microbenchmark twice. In the base case, each of these registers has the same address, making all loads cache hits (after the first iteration loads the caches). This allows us to measure any delays when a load is immediately followed by a use, since we know the number of cycles needed to execute cache hits, add instructions, and branch-on-counter instructions in the 405 core; any extra cycles in iteration time are due to load/use timing restrictions. To determine the time for a cache miss, we note that the 405GP data cache is two-way set-associative. By setting `r3`, `r4`, and `r5` to be different addresses in the same cache set, we can force a cache miss for each load. The third load evicts the first load from the cache. Looping back around, the first load then evicts the second load, the second load evicts the third, and so on. The loop is executed ten million times. The time difference between the base case and the cache-miss case is the time for thirty million cache misses, allowing the time for one cache miss to be determined. In Section 4, we show how this microbenchmark may be used for power modeling.

The microbenchmark tests were timed on the PowerPC 405GP-based Pecan board and studied with the aid of the IBM RISCTrace* tools to discover the actual behavior of the core under certain conditions. Most of this timing validation effort revolved around memory operations, the store buffer, and cache units.

Validation methodology

To validate the timing accuracy of our simulator, we used Version 1.0 of the EEMBC [10] benchmark suite.[1]

[1] The EEMBC benchmarks were used only for validation purposes, with permission. Performance and power results reported here may not comply with official EEMBC reporting guidelines and may not represent the performance of those benchmarks on the PowerPC 405GP.

We ported the benchmarks to run on the Pecan board operating system and created two boot ROM images, one for the hardware and the other for the simulator. The simulator image differs from the hardware image because the Mambo version of the operating system does not support some hardware devices. All of the benchmarks and their datasets were also included in the ROM. On both platforms (hardware and Mambo), we used the time base counters, programmed to use the internal CPU clock, to measure execution time for each benchmark. In addition, before each benchmark started, we disabled interrupts and address translation to avoid any interaction with the operating system. We ran all iteration-based benchmarks for 50 iterations and used all available datasets for those that had such options.

Timing results

Table 2 summarizes the timing validation results. For each benchmark, the table indicates the number of cycles required to execute the benchmark both on the hardware and on Tempo. The last column shows the simulated execution time error. The average error across all benchmarks was 0.6%, with a standard deviation of 2.5%. These results strongly support the accuracy of the timing information provided by Tempo.

4. Power model and validation

Achieving good cycle accuracy was the first step in our implementation because of the time dependence between energy and power. This section describes the methodology we followed to generate the event-based power model used in Tempo and our validation results.

Event-based power model

Our power model assumes that the total energy consumed by a processor is the sum of its static (or idle) energy consumption and the energies consumed by the pipeline, execution units, and caches as they execute various microarchitectural events. Thus, given a static energy E_{idle} and a set of events i with energies e_i, the total energy E_{total} consumed by a given application is given by

$$E_{\text{total}} = E_{\text{idle}} + \sum_{\text{All } i} (e_i \times n_i),$$

where n_i denotes the number of executed events of event type i. In Tempo, this energy is computed on a per-cycle basis, allowing us to produce a cycle-by-cycle power-consumption graph of the processor core. Unfortunately, the 405GP does not include hardware event monitors, which increased the complexity of our task. Furthermore,

Table 2 Summary of timing validation results.

Benchmark	*No. of hardware cycles*	*No. of simulated cycles*	*Error (%)*
a2time	413,223	426,739	3.3
aifftr	144,395,655	148,928,991	3.1
aifirf	740,301	741,021	0.1
aiifft	130,856,883	135,408,005	3.5
autcor (pulse)	300,555	300,207	−0.1
autcor (sine)	44,774,787	44,774,970	0.0
autcor (speech)	42,725,127	42,725,001	0.0
basefp	3,223,875	3,324,118	3.1
bezier	88,849,557	95,193,983	7.1
bitmnp	15,206,409	15,788,075	3.8
cacheb	45,441	44,292	−2.5
canrdr	38,601	37,291	−3.4
cjpeg	60,585,141	61,566,468	1.6
conven (k3)	15,241,887	15,248,499	0.0
conven (k4)	19,492,803	19,495,072	0.0
conven (k5)	22,412,505	22,418,867	0.0
dither	220,016,577	227,779,585	3.5
djpeg	52,295,116	52,653,158	0.7
fbital (pent)	55,151,595	54,218,153	−1.7
fbital (step)	5,639,745	5,577,367	−1.1
fbital (typ)	117,365,997	116,698,767	−0.6
fft (sine)	12,731,943	12,941,597	1.6
fft (spn)	12,702,291	12,912,470	1.7
fft (tpulse)	12,702,291	12,909,947	1.6
filters	253,353,351	256,038,518	1.1
idctm	17,032,503	17,145,522	0.7
iirflt	708,351	731,537	3.3
matrix	227,769,267	237,158,898	4.1
ospf	14,621,890	14,096,063	−3.6
pktflow	101,628,513	98,222,474	−3.4
pntrch	5,834,997	5,798,781	−0.6
puwmod	47,043	44,873	−4.6
rgbcmy	221,814,574	220,380,235	−0.6
rgbyiq	243,783,219	244,510,864	0.3
rotate	120,452,955	120,622,977	0.1
routelookup	43,405,971	43,824,845	1.0
tblook	1,364,169	1,442,869	5.8
ttsprk	594,351	580,387	−2.3
viterbi (gett)	45,632,349	45,297,193	−0.7
viterbi (ines)	45,643,815	45,296,846	−0.8
viterbi (toggle)	45,604,809	45,269,109	−0.7
viterbi (zeros)	45,518,001	45,160,094	−0.8
		Average error:	0.6
		Error standard deviation:	2.5
		Minimum error:	−4.6
		Maximum error:	7.1

we did not have *a priori* knowledge of the events that are of interest from the point of view of power. These questions were resolved through experimentation and educated guesses (based on our knowledge of architecture and VLSI circuits) about the sources of energy consumption in a processor.

To determine the actual energy cost of each of these events, we used a Pecan board instrumented for power measurement by means of a National Instruments[2]

[2] National Instruments Corporation, Austin, TX.

Table 3 Power modeling events.

Event	*Description*
CPU static energy	Base energy for every simulated cycle
Switching	Average energy due to switching in pipeline
NOP	NOP instruction (additional base energy in active state)
ALU	Logic, addition, subtraction, move, etc. instructions
Load/store	Load/store instructions
Divide/multiply	Divide/multiply instructions
PFB0 branch	Branch instruction placed in PFB0 buffer
DCD branch	Branch instruction placed in decode stage
Mispredicted branch	Branch misprediction (flushing pipeline)
ITLB miss, UTLB hit	ITLB miss satisfied by the UTLB
DTLB miss, UTLB hit	DTLB miss satisfied by the UTLB
TLB read	*tlbrehi* or *tlbrelo* instruction
TLB search	*tlbsx* instruction
TLB write	*tlbwehi* or *tlbwelo* instruction
TLB sync	*tlbsync* instruction
I-cache hit	I-cache hit to same cache line as before
I-cache hit other	I-cache hit to another cache line
I-cache miss	I-cache miss
D-cache hit	D-cache load/store hit to same cache line as before
D-cache hit other	D-cache load/store hit to another cache line
D-cache miss	D-cache miss
D-cache line flush	D-cache replacement causes a writeback

SCXI-1000-based data-acquisition system. Power planes on the board were separated for the various components using the same voltage levels, and the power-supply lines included sense resistors. Voltage drops across these resistors were used to compute the current dissipated by the devices under test and were sampled at 10 kHz.

We started by measuring the idle power of the processor while it was in the wait state, the processor state used by the Pecan board kernel in the idle loop. We then developed about three hundred microbenchmarks to measure the energy cost of certain processor core events. Many of the microbenchmarks used to determine the timing values required for the development of Tempo were also used to measure the energy costs of the various events of interest. For example, consider the microbenchmark from Section 3 that was used to determine the time for a cache miss. In the base case, with all three registers set to the same address, all accesses to memory are cache hits. In the "cache-miss" case, the registers are set to different addresses in the same cache set, and all accesses to memory are cache misses. If we define C_1 as the number of cycles per iteration in the base case, C_2 as the number of cycles per iteration in the cache-miss case, P_1 as the measured average power consumption for the base case, P_2 as the measured power consumption for the cache-miss case, and P_{idle} as the measured idle power, we can compute the energy consumed by an iteration $E_i = P_i T_i$ (where $T_i = C_i \times 5$ ns, since the core is running at 200 MHz). To measure the energy cost of a load miss, $E_{\text{load\_miss}}$, we used the following equation:

$$P_2 T_2 - P_1 T_1 = E_2 - E_1 = (T_2 - T_1) P_{\text{idle}} + 3E_{\text{load\_miss}} .$$

Note that in this example, loads replace only clean lines, so there are no memory flushes. In addition, since loads are followed by uses, the pipeline stalls until the loads return, hence the term involving P_{idle}. When the pipeline stalls waiting for a load miss, it will probably consume more power than the wait state, which implies that the load-miss energy cost is somewhat exaggerated. Attributing the energy cost of the stall to the load miss is also questionable, because the core can overlap load misses with the execution of independent instructions. We discuss other means for improving our power model in the next section.

Similar microbenchmarks and equations were derived for the other power events. To measure instruction-specific energy costs, we based our measurements on a loop of NOP instructions. We then added the instruction of interest to the loop and measured the contribution of that instruction/functional unit to power consumption. The final event list consisted of the average energy cost of cache hits/misses, various instruction types, branch conditions, interrupts, TLB hits/misses, etc. **Table 3** shows

the general events that we modeled to determine energy usage.

The only components of the power model that did not correspond to architectural events were those associated with the different instruction types and average instruction switch energy. The average instruction switch energy was used to reflect the energy consumed by the pipeline and control path of the processor as different instructions progress through the core. The additional overhead of tracking these events and their associated energies in the simulator was negligible, since it required only the addition of an energy value to the information structure of each instruction. The rest of the energy values were inserted in a table and referenced as appropriate during runtime.

Once all of the energy values were calculated, Tempo was modified to accumulate a total energy value every cycle. These energy-per-cycle values were accumulated and averaged to emit an average power value at specified intervals. The interval was set to 20 000 cycles to mimic the sampling rate of our measurement apparatus.[3] The Tempo power points were plotted to create a power graph such as that in Figure 3 (shown later).

Power validation

We performed two types of power validation. First, we compared the simulated total energy with the total energy measured on the Pecan board. Second, we compared the power graphs generated by Tempo with those measured from hardware.

We computed the energy consumed by the applications by summing all of the event energies generated during simulation and compared the total simulated energies to those computed by our measurement equipment. The hardware power samples were dumped to a file and integrated over the execution time of the applications to calculate the total energy consumed by each application. To synchronize the beginning of power measurement with the start of each benchmark run, we used GPIO signals on the Pecan board to trigger the measurements. We removed applications with short execution times from this process because the number of power points captured was small. Although our simulator was capable of emitting power estimates at a fine granularity, our hardware power measurement system was not capable of accurately and consistently measuring the power behavior of applications with very short execution times.

As shown in **Table 4**, the error in the energy predicted by the simulator compared with hardware ranged between −11.3% and 6.6%, with an average of −4.1% and a standard deviation of 5.1%.

While these power values are quite good, there is some variation from the measured values. Some of the factors that might contribute to the variation are the following:

1. The effect of instruction-related switching in the pipeline and control path of the processor was not modeled accurately. As we showed in Table 3, only an average instruction switching value was used. We have observed some substantial power variations when the instructions within a loop are reordered without affecting functionality or performance. For example, a loop with six loads and six independent adds consumes 15% less power if the loads are all executed consecutively followed by all of the adds compared with the same loop when alternating between loads and adds. We used an average value based on a limited number of experiments because an exhaustive set of experiments to capture all instruction ordering permutations was not practical. In addition, we examined the important loops in many applications and realized that opportunities for varying instruction scheduling to reduce power were difficult, primarily because of dependences and short basic block lengths in the codes examined.
2. A small amount of energy is also consumed depending on the Hamming distance of different register number encodings, and we do not model that [11].
3. We might have missed some event(s) that are significant for estimating power.
4. We might not have isolated events to a sufficient degree of separation or bundled events that should be broken down further into subevents that do not always occur together or in the same order.
5. There might be interactions between our events that were not modeled. For example, we did not measure the interaction between a load hit and a buffered store hit that attempt to access the data cache concurrently. In addition, we did not isolate the cost of various store buffer states, although modeling them is important from a performance perspective.

Despite the fact that there are many possible sources of the variation between our model and the energy measured by our hardware, we feel that our model is very effective in predicting the energy usage of the processor.

Besides its effectiveness at predicting energy consumption, the most important feature from our perspective (and from that of a potential user) is the ability of the simulator power model to depict transient power-consumption behavior while an application is

[3] The processor speed was 200 MHz and the sampling rate of the measurement equipment was 10 kHz. Note that 20 000 cycles corresponds to a 0.1-ms sampling interval.

Table 4 Summary of energy validation results.

Benchmark	*Simulated energy* (mJ)	*Actual energy* (mJ)	*Energy error* (%)
a2time	2.50	2.40	4.44
aifftr	807.52	819.40	−1.45
aifirf	4.19	4.36	−3.99
aiifft	734.07	752.80	−2.49
autcor (pulse)	1.62	1.70	−4.67
autcor (sine)	241.05	258.40	−6.72
autcor (speech)	230.02	245.80	−6.42
basefp	19.44	18.49	5.12
bezier	522.69	502.30	4.06
bitmnp	84.33	81.30	3.73
cjpeg	342.15	354.50	−3.48
conven (k3)	82.28	87.12	−5.55
conven (k4)	104.95	111.60	−5.96
conven (k5)	120.67	127.80	−5.58
dither	1238.39	1291.00	−4.07
djpeg	296.86	311.90	−4.82
fbital (pent)	289.11	318.50	−9.23
fbital (step)	29.82	32.89	−9.32
fbital (typ)	633.24	677.90	−6.59
fft (sine)	68.57	70.36	−2.55
fft (spn)	68.42	71.25	−3.97
fft (tpulse)	68.40	70.77	−3.34
filters	1335.55	1389.00	−3.85
idctm	91.19	92.38	−1.28
iirflt	4.17	3.99	4.48
matrix	1378.78	1298.00	6.22
ospf	80.83	88.16	−8.31
pktflow	539.48	593.30	−9.07
pntrch	31.95	33.64	−5.03
rgbcmy	1257.79	1369.00	−8.12
rgbyiq	1332.43	1394.00	−4.42
rotate	662.80	701.80	−5.56
routelookup	252.91	283.50	−10.79
tblook	8.37	7.85	6.61
ttsprk	3.15	3.30	−4.66
Viterbi (gett)	254.85	285.30	−10.67
Viterbi (ines)	254.85	286.10	−10.92
Viterbi (toggle)	254.73	287.20	−11.30
Viterbi (zeros)	254.70	284.10	−10.35
		Average error:	−4.1
		Error standard deviation:	5.1
		Minimum error:	−11.3
		Maximum error:	6.6

running. This aspect of the validation process is somewhat subjective, but to give the reader an idea of the effectiveness of the model, we have chosen to include a few illustrations. **Figures 3(a)–3(d)** show the hardware measured power during the execution time of an application and the corresponding simulator-generated curves for `cjpeg`, `fft.sine`, `matrix`, and `djpeg`, respectively. These applications were chosen because they have distinct power variations during runtime. The simulator is capable of depicting most of this variation in power consumption for the applications, but there are some interesting deviations that we are attempting to narrow.

For example, for `cjpeg`, the relative power during the initialization phase (the first six steps in the curve) was inverted compared with that of the hardware. We are also

trying to understand the reason for the apparent power offset in many of the curves, accounting for a large portion of the error. We suspect that it is due to the use of wait state power as the base. Intuitively, the base power while the processor is idle, but out of the wait state, should be slightly higher. Again, the consistency of our power prediction relative to hardware measurements suggests the usefulness of our event-based energy model for studying the performance and power behavior of software running on the 405GP.

Power profiling—A user's perspective

After completion of the timing and power validation effort, we implemented a prototype interface to assist users in understanding the sources of power consumption in their applications. We created a graphical user interface (shown in **Figure 4**) that displays, for an application running on Mambo, a real-time graph of the power consumption broken down into some of its subparts (e.g., power due to cache misses, functional units, etc.). In addition, the graphical user interface allows users to select a section of the graph that is of interest and display the simulated code responsible for that behavior. This interface can easily be extended to provide graphical representations of many events of interest by using the Mambo emitter interface, which allows the simulator to generate events that are processed by a separate application in real time. The simulator can also display a breakdown of many events that occur within an application, along with typical performance statistics.

5. Related work

There is a substantial amount of related work in the area of power modeling; we limit our discussion to the most closely related efforts. Wattch [4] is a power-simulation tool that estimates the power consumption of various processor structures by including efficient power models that use measurements of activity levels to arrive at power estimates. These power models are usually tailored for a specific technology. Wattch has been successfully integrated into architectural simulators such as SimpleScalar [12]. Our approach differs from theirs because we use lookup tables, resulting in even lower runtime overhead. In addition, our model has been validated on real hardware. On the other hand, when Wattch models are accurate, they provide a better architecture investigation tool because they can predict the power consumed by resized structures without having to regenerate the power model, which is the case with Tempo. However, even the Wattch power model must be recalibrated when it is applied to a different semiconductor fabrication technology.

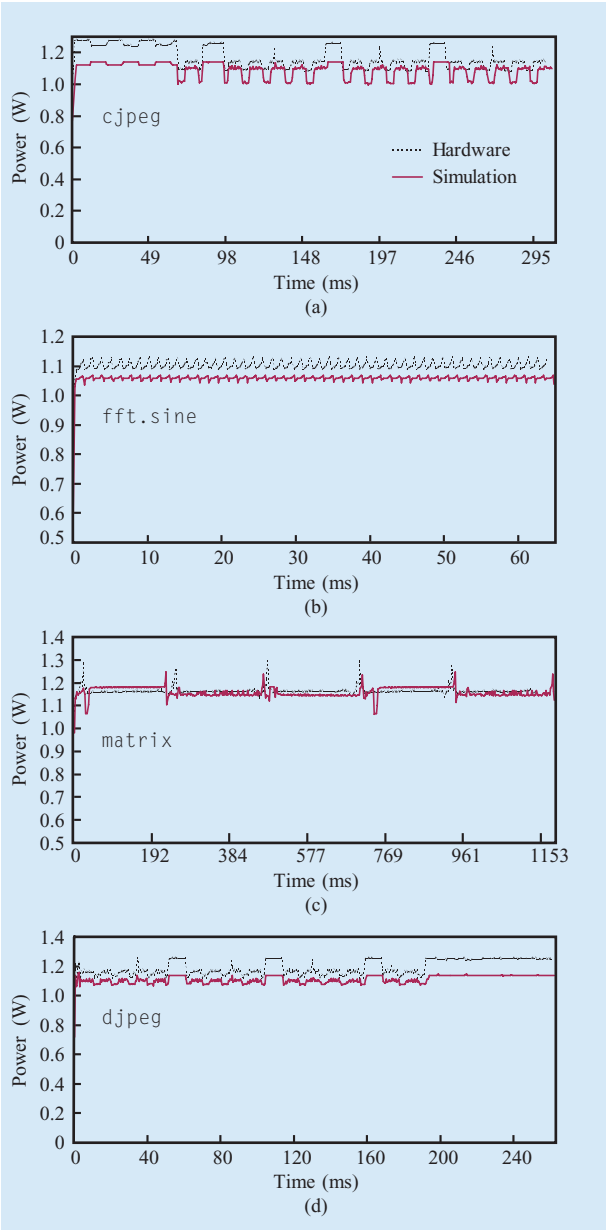

Figure 3

Comparison of hardware and simulated power graphs for (a) `cjpeg`, (b) `fft.sine`, (c) `matrix`, and (d) `djpeg`.

Event-based energy estimation has previously been proposed for use on systems that include hardware event counters [5, 13]. That approach is useful for average power-consumption estimates across coarse-grained portions of applications because it requires reading hardware performance monitors. Our simulator does not suffer from this granularity problem and can use a larger set of events to estimate transient power behavior. Also,

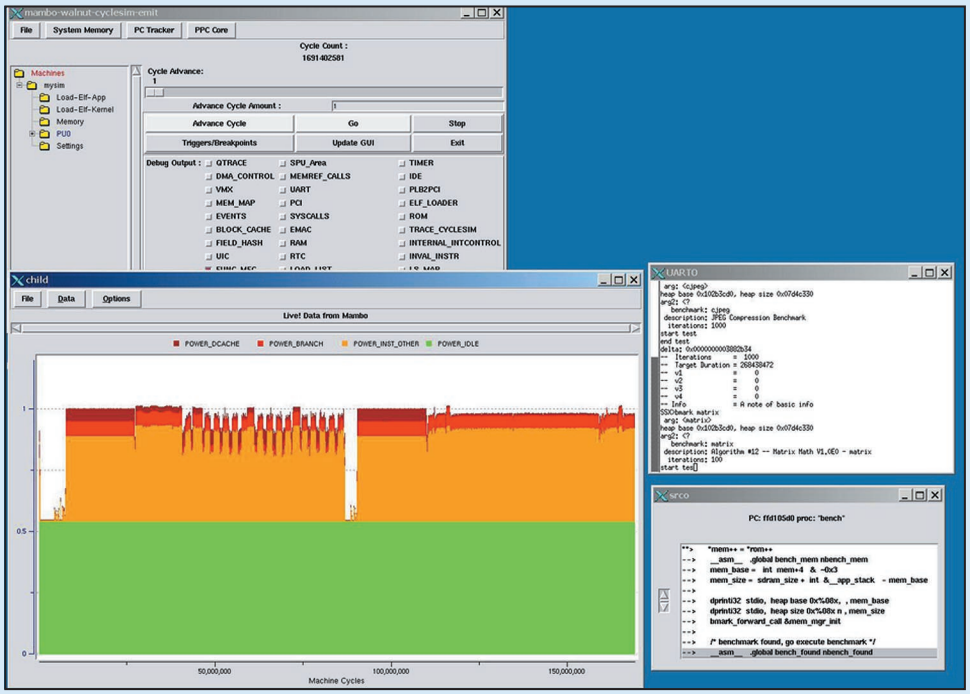

Figure 4

Mambo power profiler interface.

our measurement-based profiling of instruction energy costs is similar to the work performed in [14].

6. Concluding remarks

In this paper, we have described the design and validation of Tempo, a PowerPC system-level performance and power simulator that should be of benefit to both architects and software developers. For existing systems, the simulator provides feedback on performance and power consumption that is very difficult and/or time-consuming to attain otherwise. For early design stage applications, we believe that our simulation methodology can leverage the many existing power-estimation tools to create an event-based power model for studying the performance and power consumption of full applications and operating systems—quickly and with a high degree of accuracy when integrated with cycle-accurate performance simulators.

We are currently involved in multiple research efforts that utilize our simulator experience for power-aware systems. In addition, we plan to model more complex microprocessors and multiprocessor systems while enhancing our I/O power modeling capabilities. Our effort to model and measure the power consumption of the 405GP SDRAM memory subsystem is nearing its completion and will be integrated into future versions of the Tempo simulator. In addition, we are generating a power model of the PowerPC 405LP processor [15], including support for dynamic voltage and frequency scaling, for use in the simulator. Finally, we are developing a performance and power model of a version of the PowerPC 750 processor.

Acknowledgments

We thank Bishop Brock for assisting us during many phases of this project, especially with the Pecan board and its kernel. We are very grateful to Chandler McDowell for his many hours of help in configuring, debugging, and calibrating our power measurement instruments. We also thank Rabi Mahapatra for his help with the power measurements. This work was partially supported by DARPA (Defense Advanced Research Projects Agency, U.S. Department of Defense) Power Aware Computing/Communication Contract F33615-00-C-1736 under Subcontract Agreement 400525-1.

**Trademark or registered trademark of Sequence Design, Inc., Synopsys, Inc., Linus Torvalds, Advanced Micro Devices, Inc., or Red Hat, Inc.

References

1. L. W. Nagel and D. O. Pederson, "Simulation Program with Integrated Circuit Emphasis," *Memorandum ERL-M382*, University of California at Berkeley, California, April 1973.
2. Sequence Design Inc., "PowerTheater Datasheet," Santa Clara, CA; see *http://www.sequencedesign.com/2_solutions/powertheater3.pdf*, 2002.
3. Synopsys Inc., "PowerMill Datasheet," Mountain View, CA.; see *http://www.synopsys.com/products/etg/powermill_ds.html*, 2002.
4. D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 2000, pp. 83–94.
5. F. Bellosa, "The Case for Event-Driven Energy Accounting," *Technical Report TR-I4-01*, Friedrich Alexander Universitat Erlanger-Nurnberg, June 2001.
6. IBM Corporation, *PowerPC 405GP Embedded Processor User's Manual*. 9th Preliminary ed., 2001; see *http://www.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_405GP_Embedded_Processor/*.
7. A. R. Weiss, "The Standardization of Embedded Benchmarking: Pitfalls and Opportunities," *Proceedings of the IEEE International Conference on Computer Design*, October 1999, pp. 492–498.
8. M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod, "Using the SimOS Machine Simulator to Study Complex Computer Systems," *ACM Trans. Modeling & Computer Simulation* **7,** 78–103 (1997).
9. G. R. Ganger, "System-Oriented Evaluation of I/O Subsystem Performance," Ph.D. Dissertation, University of Michigan, Ann Arbor, June 1995.
10. Embedded Microprocessor Benchmark Consortium, "EEMBC Technical and Operating Specification; see *www.eembc.org/*, 2002.
11. S. Lee, A. Ermedahl, and S. L. Min, "An Accurate Instruction-Level Energy Consumption Model for Embedded RISC Processors," *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems* (*LCTES*), 2001, pp. 1–10.
12. D. Burger and T. Austin, "The SimpleScalar Tool Set, Version 2.0," *Computer Architecture News*, pp. 13–25 (1997).
13. R. Joseph and M. Martonosi, "Run-Time Power Estimation in High-Performance Microprocessors," *Proceedings of the International Symposium on Low Power Electronics and Design* (*ISLPED*), August 2001.
14. V. Tiwari, S. Malik, and A. Wolfe, "Instruction Level Power Analysis and Optimization of Software," *J. VLSI Signal Process. Syst.* **13,** 223–233 (1996).
15. IBM Corporation, *PowerPC 405LP Embedded Processor*, 2003; see *http://www.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_405LP_Embedded_Processor/*.

Received November 10, 2002; accepted for publication March 20, 2003

Hazim Shafi *IBM Research Division, Austin Research Laboratory, 11501 Burnet Road, Austin, Texas 78758 (hshafi@us.ibm.com).* Dr. Shafi received a Ph.D. degree from Rice University in 2000. While at Rice, he was a member of the Rice Simulator for ILP Multiprocessors group, in which he worked on techniques to reduce and tolerate memory latency in ccNUMA systems. He later joined the Brazos software distributed shared-memory project at Rice, in which he worked on fault tolerance and cluster management research using checkpoints and thread migration. Since joining IBM in 2001, Dr. Shafi has developed a cycle-accurate, power-aware simulator within the Mambo simulation environment. He is currently investigating performance evaluation techniques for future large-scale shared-memory systems; he is the principal investigator for the DARPA PAC/C project activities at the Austin Research Laboratory.

Patrick J. Bohrer *IBM Research Division, Austin Research Laboratory, 11501 Burnet Road, Austin, Texas 78758 (pbohrer@us.ibm.com).* Mr. Bohrer is a Senior Software Engineer in the Austin Research Laboratory. He received a B.S. degree in computer science from the University of Texas at Austin in 1990 and then joined Motorola in Austin, Texas. He moved to AT&T Bell Laboratories in Allentown, Pennsylvania, in 1993 and then joined IBM in 1994. Mr. Bohrer has worked on a variety of compilers, tools, and simulators to support the Motorola 88110, AT&T Hobbit, and PowerPC architectures.

James Phelan *IBM Software Group, 11501 Burnet Road, Austin, Texas 78758 (jmphelan@us.ibm.com).* Dr. Phelan received a Ph.D. degree in computer science from New Mexico State University in 1988. He had previously worked for the Department of Defense and NASA as a physicist. He joined IBM in 1991 and has worked on various operating system and performance projects since then.

Cosmin A. Rusu *Computer Science Department, University of Pittsburgh, Sennott Square, Pittsburgh, Pennsylvania 15260 (rusu@cs.pitt.edu).* Mr. Rusu received a B.S. degree in computer science from the Technical University of Cluj-Napoca, Romania, in 2000. He joined the University of Pittsburgh that same year and is currently pursuing a Ph.D. degree in the Computer Science Department. His research interests include real-time systems and power-constrained systems.

James L. Peterson *IBM Research Division, Austin Research Laboratory, 11501 Burnet Road, Austin, Texas 78758 (petersjl@us.ibm.com).* Dr. Peterson is a member of the research staff of the IBM Austin Research Laboratory. He received a Ph.D. degree from Stanford University in 1974 and then joined the Department of Computer Sciences of the University of Texas at Austin; he joined IBM in 1989. Dr. Peterson has published four books (including the popular *Operating Systems Principles* textbook) and a number of papers on computers, software, algorithms, and systems.