Power Modeling for Heterogeneous Processors

Tahir Diop
Department of Electrical and Computer Engineering
University of Toronto
Ontario, Canada
tahir.diop@mail.utoronto.ca

Natalie Enright Jerger
Department of Electrical and Computer Engineering
University of Toronto
Ontario, Canada
enright@eecg.toronto.edu

Jason Anderson
Department of Electrical and Computer Engineering
University of Toronto
Ontario, Canada
janders@eecg.toronto.edu

ABSTRACT

As power becomes an ever more important design consideration, there is a need for accurate power models at all stages of the design process. While power models are available for CPUs and GPUs, only simple models are available for heterogeneous processors. We present a micro-benchmark-based modeling technique that can be used for chip multiprocessor (CMPs) and accelerated processing units (APUs). We use our approach to model power on an Intel Xeon CPU and an AMD Fusion heterogeneous processor. The resulting error rate for the Xeon's model is below 3% and is only 7% for the Fusion. We also present a method to reduce the number of benchmarks required to create these models. Instead of running micro-benchmarks for every combination of factors (e.g. different operations or memory access patterns), we cluster similar micro-benchmarks to avoid unnecessary simulations. We show that it is possible to eliminate as many as 93% of the compute micro-benchmarks, while still producing power models having less than 10% error rate.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Heterogeneous (hybrid) systems; I.3.1 [**Computer Graphics**]: Hardware Architecture; I.6.5 [**Simulation and modeling**]: Modeling methodologies

General Terms

Experimentation, Measurement, Performance

Keywords

Power modeling, Benchmarking, Statistical analysis

1. INTRODUCTION

Modern processors have hit a power density limit, which has stalled increases in clock frequencies, led to the emergence of heterogeneous processors, and raised the importance of power in computer architecture [1]. For architects to be able to give power the attention it deserves, they need to be able to evaluate power throughout the design process. This work presents bottom-up power models for an Intel Xeon CPU and an AMD Fusion APU – the first such model for a heterogeneous processor. The APU model takes into account the power consumption of the CPU, GPU, and shared resources. The proposed models are built using power measurements of real hardware, performance information from the Multi2Sim simulator [2], and hardware performance monitoring counters (PMCs).

Bottom-up modeling makes use of micro-benchmarks to find the link between power consumption and specific components of a processor. For an accurate model, we want these components to be used in as wide a range of conditions as possible, leading to large numbers of micro-benchmarks. Since we use not only hardware measurements, but also simulation, the number of micro-benchmarks is an impediment to rapid power model development. To reduce the amount of simulation time required, we present a methodology that can determine, based on power measurements, which micro-benchmarks are truly important for the modeling process. We use micro-benchmark clustering based on total energy consumption to determine which micro-benchmarks are similar enough to others that they can be eliminated from the modeling process.

Figure 1 shows the entire modeling process. Models are developed based on power measurements made with the setup described in Section 3.1, as well as PMC measurements on real hardware, described in Section 3.2, and simulating the workloads in Multi2Sim for performance information, described in Section 3.3. For the benchmark selection process, the benchmark set developed in Section 4 was first used. Modeling was also performed with reduced benchmark sets, as described in Section 5. The modeling process itself is described in Section 6. Finally, the different models are validated and compared in Section 7.

2. ARCHITECTURE BACKGROUND

Power models were generated for two different architectures: a CMP to demonstrate the effectiveness of the modeling technique, and a heterogeneous processor to show that the methodology can be extended to more complex architectures. Both target architectures are described below.

2.1 Xeon CPU

The conventional CPU used in this work is the four core Intel Xeon W3530, based on the Bloomfield architecture [3].

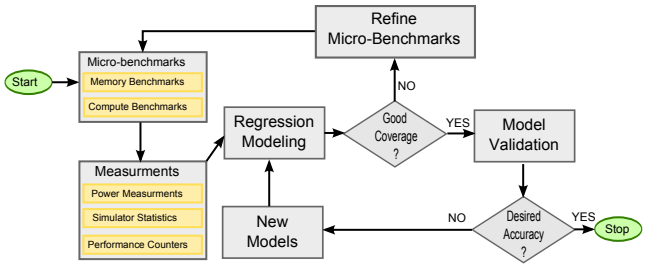

Figure 1: Steps involved in the modeling process.

Table 1: Intel Xeon W3530 Specification.

Component	Value
CPU cores	4
CPU architecture	Bloomfield
CPU Operating Frequency	2.8 GHz
L1 I/D-cache	32 kB per core
L2 Cache	256 kB per core
L3 Cache	8 MB shared
Memory controller	Triple channel DDR3
TDP	130 W

This processor includes three levels of cache and an integrated memory controller. Each core has private 32 kB L1 instruction and data caches, as well as a private 256 kB unified L2 cache. There is also an 8 MB L3 cache shared by all four cores. The L3 is an inclusive design, meaning that any value contained in a core's private cache is also contained in the L3 cache. The Xeon's specifications are summarized in Table 1.

2.2 Fusion APU

The heterogeneous processor used in this work is the AMD Fusion A6-3650, a Llano APU [4] with: four CPU cores, four GPU cores, and a shared memory controller. The CPU is based on the family 12h [5] architecture and the GPU is based on the Evergreen architecture [6]. The APU's specifications are summarized in Table 2. The A6-3650's CPU includes two levels of cache and an integrated memory controller shared with the GPU. Each core has private 64 kB L1 instruction and data caches, as well as a private 1 MB unified L2 cache. The caches are exclusive. The GPU in the APU uses a very long instruction word (VLIW) instruction set architecture. The GPU contains texture caches, but they are cannot be used for GPGPU computation.

Table 2: AMD A6-3650 Specification.

Component	Value
CPU cores	4
CPU architecture	family 12h
CPU Operating Frequency	2.6 GHz
L1 I/D-cache	64 kB per core
L2 Cache	1 MB per core
GPU cores	4
GPU architecture	Evergreen
GPU Operating Frequency	443 MHz
Streaming Processors	16 per core
Stream Processing Unit	5 per-streaming processor
Local memory	32 kB per core
Memory controller	Dual channel DDR3
TDP	100 W

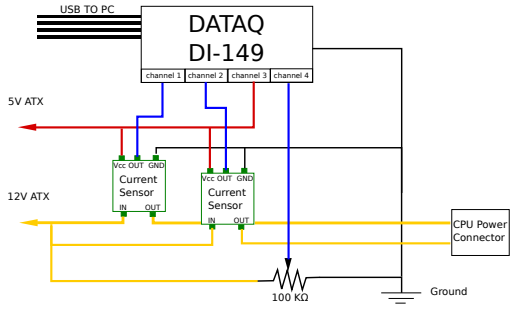

Figure 2: Schematic of the measuring setup.

3. MEASUREMENT SETUP

3.1 Power Measurement

Power was measured at the package level at normal operating temperatures. Benchmark execution time was kept low and the processor was allowed to idle between benchmarks. Measured values include the power consumption of the CPU and memory controller for the Intel CPU, and also include the GPU for the AMD APU.

To measure the power consumption, both the current and voltage delivered to the package were measured. Measurements were made using DataQ's DI-149 [7] data acquisition (DAQ) unit. It can measure differential voltage up to 10 V at a maximum sampling rate of 10 kHz.

Four channels were used to measure the chip's power consumption: two were used to measure current, one to measure voltage on the 12 V line, and another to measure voltage on the 5 V line. Current was measured by inserting current sensors on the 12 V lines of the power connector between the power supply and motherboard. In accordance with the ATX power specification [8], only this connector delivers power to the CPU/APU's voltage regulators. We used Allegro's ACS711 current sensors [9]. To measure voltage across the 12 V line, a 100 kΩ potentiometer was used to halve the voltage. A sampling rate of 1 kHz or 250 Hz per-channel was used. Figure 2 shows a schematic of the measurement setup.

To reduce the impact of other concurrently running applications contributing to the power measured, a system with a clean installation of Ubuntu 12.04 LTS was used. The system was run without a display, so the GPU would have no additional tasks, and an SSH connection without X forwarding was used to access the system. Cool'n'Quiet [10] and SpeedStep [11], AMD's and Intel's respective P-state [12] dynamic voltage and frequency scaling (DVFS) implementations were disabled, meaning that the processors were always operating at their maximum frequencies. However, C-states [12] were not disabled, so idle CPU or GPU cores can still be gated.

3.2 Performance Counters

Modern processors contain performance monitoring counters (PMCs) that can be programmed to monitor certain hardware events. Using such counters has very low overhead [13]. The counters of interest for this work are those related to memory operations, specifically the following:

1. **Requests to L2 Cache** – which counts the number of L2 requests from L1 instruction and data caches.
2. **L2 Cache Misses**
3. **DRAM Accesses** – which counts the number of read and write requests to the DRAM (on the Fusion only).
4. **L3 Cache Misses** – which counts the number of L3 cache misses (on the Xeon only).

By measuring L2 cache misses and DRAM accesses separately on the Fusion, it is possible to distinguish between cache misses that could be satisfied by other cores from those that were satisfied by DRAM. Since the Xeon has a single shared inclusive L3 cache, all L2 cache misses are forwarded to the L3. L3 cache misses can only be satisfied by the main memory. We also count of the number of memory requests made by the CPU during simulation with Multi2Sim. With these four values we can determine the number of hits and misses to each level of cache.

While PMCs provide a quick and low overhead way of gathering performance data they do have certain limitations. The number of counters is limited to four on both processors, meaning that counts from multiple benchmarking runs would need to be combined. However, the biggest limitation is that they provide little insight into the execution pipeline. For example, we have no way of separating an integer divide from a multiplication. Therefore, we chose to use a simulator to gather the rest of the performance information. This allows us to get detailed information about what is taking place in each processor pipeline during benchmark execution.

3.3 Multi2Sim Simulator

Multi2Sim [2] is an open source heterogeneous architecture simulator, capable of performing cycle-accurate simulations for both CPUs and GPUs. Multi2Sim is the only GPGPU simulator that simulates AMD GPUs and specifically, the Evergreen micro-architecture. While the x86 CPU model in Multi2Sim is not based on any existing hardware, it is highly configurable. This makes Multi2Sim an ideal platform for simulating the Fusion.

However, Multi2sim exhibits a few limitations that were discovered over the course of this work. Not all instructions are implemented in the Evergreen model, meaning that not all application benchmarks could be simulated for the GPU. Also, Multi2Sim simulates an inclusive cache hierarchy, while the Fusion CPU has an exclusive cache, and memory accesses are modeled with a constant latency. However, since all our cache and memory data is obtained from PMCs, this limitation does not affect any simulation statistics used in our modeling.

4. BENCHMARKS

It is necessary to ensure that the benchmarks used to create a power model represent a wide enough range of workloads for the resulting model to be valid. For example, if no workloads include multiplications, we cannot expect to model the energy consumption of a multiplication accurately. Furthermore, we need to ensure that certain types of workloads are not overrepresented. Micro-benchmarks are used to achieve these goals and target all components of the architecture. To ensure the importance of individual components is not exaggerated, application benchmarks that make use of a wide range of components are also used, to ensure that any constant energy consumption (e.g. leakage current) is not associated with any of our performance statistics.

4.1 Micro-Benchmarks

A micro-benchmark exercises one specific component of a processor. By using a diverse set of micro-benchmarks, it becomes possible to characterize the entire processor. This can be done for performance and power.

4.1.1 Memory Micro-Benchmarks

The GPU's memory is fairly simple to test, because there are only two type of memory: main memory and per-core local memory. Main memory supports contiguous accesses of up to 256 bits. To achieve maximum memory bandwidth, applications should make contiguous memory accesses.

Three types of main memory accesses were considered: contiguous, sparse, and conflict. For the contiguous case, each core makes contiguous reads or writes to separate regions of memory allowing for maximum throughput. For the sparse case, threads on a core access memory with a stride of 256 bits. This prevents the coalescing of accesses from a single core. For the conflict case, every thread on a core attempts to read or write the same memory location. This forces the writes to be serialized and reduces the amount of data transferred for reads.

Two types of local memory access were also considered: contiguous and conflict. There was no need for a sparse access micro-benchmark as there is no penalty for non-contiguous local memory accesses, only for conflicts.

The CPU's memory is more complex for two reasons. First, with main memory and multiple levels of cache there are more types of memory to consider. Second, since the caches are hardware-managed, we require lower level micro-benchmarks to ensure we produce the desired behavior. We created an additional micro-benchmark that allows accesses to be made to each level of the memory hierarchy separately. This allows us to control the number of memory accesses to the various cache levels.

4.1.2 Compute Micro-Benchmarks

For the compute micro-benchmarks, the following factors were considered: the operation, the data-type, the data values of the operands, the instruction level parallelism (ILP), and the number of cores used. The possible values for each of these parameters are given in Table 3. The operations used include: arithmetic, relational, and bitwise operators. If the ILP is one, operations will use the result of the previous operation as an operand. On the CPU, this implies that only one pipeline can be active at a time. On the GPU, this implies that multiple operations cannot be combined into a single VLIW instruction. If the ILP is five, then operations will use the result of the fifth-most recent operation as an input. The value of five was chosen because Evergreen is a VLIW-5 architecture, while the CPU is only 3-wide. By

Table 3: Possible Factor Values for Micro-Benchmarks.

Parameter	Possible Values
Operation	+, −, ∗, /, &, |, ∧, <<, ∼, ++, <, !=, int/float conversion
Data-types	int, int4, float, float4
Input data Values	0000, 1111, 0101, ascending values
ILP	1, 5
Cores	1, 2, 3, 4

Table 4: Applications Benchmarks Used

Application name	Source	CPU	GPU	APU
Black Scholes (C/x87))	PARSEC	x		
Bodytrack	PARSEC	x		
Canneal	PARSEC	x		
Dedup	PARSEC	x		
Fluidanimate	PARSEC	x		
Swaptions	PARSEC	x		
Binomial Option	APPSDK	x		
Black Scholes (OpenCL/SSE)	APPSDK	x		
DCT	APPSDK	v		
N-body	APPSDK	x		
Scan large array	APPSDK	x		
Mandelbrot	APPSDK	v	v	
Matrix multiplication	APPSDK	v	v	
Matrix multiplication (local memory)	APPSDK	x	v	
Matrix transpose	APPSDK	x	x	
Prefix sum	APPSDK		x	
Reduce	APPSDK	x	x	
Sobel Filter	APPSDK	x	x	
URNG	APPSDK	x	x	
Vector	DistCL [17]	v	v	
K-means	Rodinia			v
LU-decomposition	Rodinia			v
Needleman-Wunsch	Rodinia			v
Pathfinder	Rodinia			v

varying the ILP, we can measure power consumption at minimal and peak architectural efficiency for both the CPU and GPU.

The same compute micro-benchmarks were used for the GPU and CPUs. However, two versions of the scalar floating point micro-benchmarks were created for the CPU. One that executes in the x87 pipeline and another that uses the SSE pipeline.

4.2 Application Benchmarks

Application benchmarks are used to simulate a mixed workload that includes a range of memory and ALU operations. It is important to include some general benchmarks that use more than one type of operation to ensure that the energy consumption of individual operations is not overestimated. When a single operation is used, its modeled cost includes not only the cost to execute the operation, but also any overhead that is not captured by another metric. Application benchmarks are also ideal for model validation, as they cover a broad set of modeled parameters. The application benchmarks had three primary sources: the AMD APPSDK v2.7 [14], the PARSEC benchmark suite [15], and the Rodinia heterogeneous benchmark suite [16]. In total, twenty-four applications were used (see Table 4).

For the CPU and GPU models, four benchmarks were used as validation benchmarks. They are marked with a 'v' in the table, and are used to assess the predictive power of the models. They are not part of the set of benchmarks used to train models. The validation sets for the CPU and GPU consist of four benchmarks each.

The PARSEC benchmarks are x86 applications and could therefore only be run on the CPU. They were added because the initial CPU models had very poor predictive ability. We include all PARSEC benchmarks that would simulate in the 48-hour time window permitted for jobs on our compute cluster. These benchmarks were run using one to four cores.

The Rodinia benchmarks consist of heterogeneous workloads that run OpenCL kernels on the GPU and perform the remaining computations on the CPU. These benchmarks were used to validate the APU power model as they target both devices, allowing us to test the predictive power of the final CPU+GPU model with benchmarks that make use of both devices. The four Rodinia benchmarks used are the ones that would simulate on the GPU.

5. BENCHMARK SET REDUCTION

In all, 1216 CPU and 928 GPU micro-benchmarks were considered. Collecting power measurement for each micro-benchmark on real hardware only takes a few hours. Simulation is much more time consuming with each CPU micro-benchmark require between 24 and 48 hours to complete. We require more than three CPU years of total simulations time. However, many of these micro-benchmarks have similar characteristics and they may not all be required for modeling. In order to demonstrate this, we require a quantitative approach to clustering micro-benchmarks, and then analyzing the commonalities in a cluster.

We used Gaussian-means (G-means) [18] clustering to group micro-benchmarks based on their commonalities. G-means is a variant of the well-known k-means clustering [19]. Unlike k-means, the G-means algorithm is capable of determining what the appropriate number of clusters for a data-set is. It starts by selecting a k and running the normal k-means algorithm. Then, each cluster is tested to determine if it follows a Gaussian, or normal, distribution. If any clusters do not appear to be Gaussian, k is incremented and the process is started again. G-means clusters the data into the minimum number of representative clusters. We used Anderson and Darling's A^2 statistic [20] to test the distribution of each cluster, as it was shown to be a powerful test with low computational complexity [21]. To increase the confidence in the clustering generated and since we have no knowledge of probable centers, we ran k-means with 50 random starts.

The G-means algorithm was applied to the data gathered from running the compute micro-benchmarks on real hardware. For each micro-benchmark we considered the factors shown in Table 3, as well as average power consumption, runtime, and total energy consumption. To allow for similar ALU operations, such as addition and subtraction, to be clustered, five operation groups were created. The groupings are listed in Table 5. This was done because from a hardware perspective, there is very little difference between an addition and a subtraction, or between any bitwise oper-

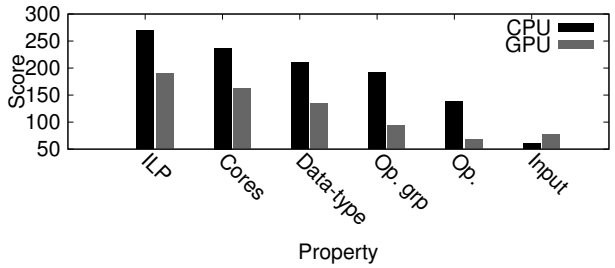

Figure 3: Property scores for the Fusion APU.

Table 5: Operation Groupings

Group	Operations Included
Logic	and, or, xor
Unary	not, increment
Comparison	less than, not equal
Arithmetic	add, subtract

ations. The micro-benchmarks were clustered based on the total energy required to run them. Energy was used because the goal is to develop an energy per-instruction (EPI) model and it combines both power consumption and runtime. Each micro-benchmark executes an operation a fixed number of times (48 000 000 in our case) making it possible to extract per-operation energy consumption.

We created two reduced micro-benchmark sets based on the G-means results. For the first, we used G-means to determine which factors have the greatest effect on energy consumption, and then vary only these factors. For each cluster we rank the properties in order and assign the most important property a score of five and the least important a score of one. The scores for each property are then summed for all clusters to determine the overall importance of each property. Figure 3 shows the property scores for the APU. For all devices, the most important factors were ILP, core count, operations group, and data-type. This resulted in the same reduced set of 184 micro-benchmarks for both the CPUs and the GPU. The next approach was to select a single micro-benchmark to represent each cluster – we call this the minimal set. This set consists of 117 micro-benchmarks for the Xeon CPU, 85 micro-benchmarks for the Fusion's CPU, and 32 micro-benchmarks for the Fusion's GPU, reductions of 90%, 93%, and 97% respectively.

6. MODELING

There are two common approaches to generating power models for processors: circuit-level modeling and statistical modeling using performance data. Circuit-level modeling is a more accurate but much more computationally expensive approach and it is used in the popular Wattch CPU power modeling framework [22]. It requires many details about the architecture and process technology to produce accurate results and can be quite burdensome in the early stage of design when these details may not yet be known. On the other hand, statistical modeling has a high up-front cost while the model is being trained, but is much faster to use. There are two common approaches to producing statistical power models: top-down and bottom-up [23]. The top-down approach treats the hardware like a black box, where no details about the modeled system need to be known. While the top-down approach is simple and yields comparable results to the bottom-up approach, it has the limitation of not being composable. With the bottom-up approach, the total power consumption can be decomposed into the power consumption of various functional units, plus any fixed consumption. This makes the bottom-up approach a good fit for power models that will be used inside an architectural simulator, and are therefore the types of models we use.

We used non-negative least squares (nnls) linear regression [24] to solve our system of equations. We have one equation per benchmark. For each equation, the dependent variable is the total energy consumption, the independent variables are the runtime of a benchmark and the counts of hardware activity, and we are trying to solve for the energy consumption of each predictor. Predictors are either statistics obtained from the simulator or the PMCs, where examples include: cycle counts, number of integer instructions, number of local memory accesses, and number of L1 cache accesses, etc. The values of the predictors vary according to the properties of the benchmarks. For example if we are doing integer addition on the GPU increasing the ILP from one five will approximately reduce the number add instructions by four fifths. Since the problem is overdetermined by design, there are more equations then unknowns, there is likely no exact solution. Instead, a regression aims to find an approximate solution by reducing the sum of squares of the residuals. The residual for each benchmark is the difference between the predicted energy consumption of a benchmark and the true energy consumption of the benchmark. The final result is an equation of the form: $E = E_{k_1} n_{k_1} + E_{k_2} n_{k_2} + ... + E_{k_m} n_{k_m} + b$ where, E is the dependent variable (total energy), E_{k_i} and n_{k_i} are the coefficient and activity count of predictor k_i, and b is the intercept (static power $\times$ runtime).

The basic problem formulation for device power is expressed in Equation 1. Total power (P) is the sum of static power (P_{stat}) and dynamic power (P_{dyn}). P_{dyn} can be further decomposed into the dynamic power of each core (P_{dyn_c}), as shown in Equation 2. As shown in Equation 3, P_{dyn_c} is the sum of dynamic power for each predictor. The dynamic power of each predictor is the product of the activity count of the predictor (n_k) and the energy consumption of the predictor (E_k) divided by time. P_{stat} can also be decomposed into the static power of each core (P_{stat_c}) and the static power of shared resources (P_{stat_S}), as shown in Equation 4. The complete breakdown of power consumption is shown in Equation 5. To generate a power model, we need to solve for P_{stat} and the E_k's. P is measured for real hardware and core count and the value of n_k are obtained from simulation. Since power is energy over time ($P = \frac{E}{t}$), it is possible to formulate the problem in terms of energy, as shown in Equation 6.

$$P = P_{stat} + P_{dyn} \tag{1}$$

$$P_{dyn} = \sum_{c=1}^{\#cores} P_{dyn_c} \tag{2}$$

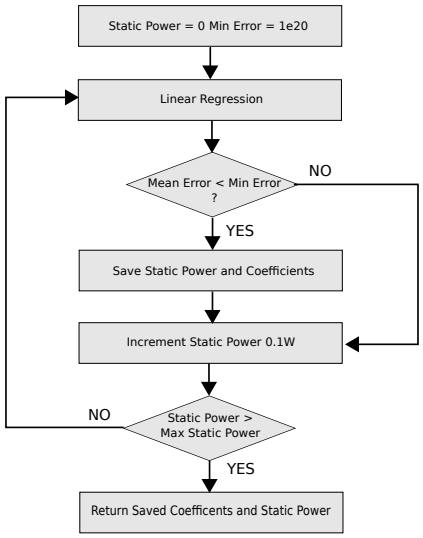

Figure 4: Regression process.

$$P_{dyn_c} = \sum_{k=1}^{\#preds.} \frac{n_k \times E_k}{t} \tag{3}$$

$$P_{stat} = P_{stat_S} + \sum_{c=1}^{\#cores} P_{stat_c} \tag{4}$$

$$P = P_{stat_S} + \sum_{c=1}^{\#cores} \left(P_{stat_c} + \sum_{k=1}^{\#preds.} \frac{n_{ck} \times E_k}{t} \right) \tag{5}$$

$$E = tP_{stat_S} + \sum_{c=1}^{\#cores} \left(tP_{stat_c} + \sum_{k=1}^{\#preds.} n_{ck} \times E_k \right) \tag{6}$$

A bottom-up approach was used to obtain a decomposable model [23]. This required the CPU and GPU power models to be developed separately for the APU. In total, 16 CPU models and 26 GPU models were considered. The complexity in models varied significantly.

The modeling process consists of solving Equation 6. What distinguishes the set equations for the various models are the $k's$ being considered. For each model, the process shown in Figure 4 was followed. The nnls solver used [25] does not support the inclusion of an intercept term in the problem formulation. Therefore, we have to perform multiple regressions assuming different static power levels. We used values of P_{stat_S} between 0 and 35W in increments of 0.1W. The maximum P_{stat_S} values are based on the range of power consumption seen in CPU and GPU benchmarks. To solve for P_{stat_c}, we added a parameter that is the product of the number of cores used, and the duration of the benchmark.

Each model is assessed by its error rate. The model is used to predict the energy consumption of each validation benchmark. Then, the residual is calculated and the relative error is found, it is the absolute value of the residual divided by the energy consumption of the benchmarks. Finally, the arithmetic mean of all the relative errors is calculated. We use an arithmetic mean instead of a geometric mean because we want to increase the cost of high error outliers and reduce the benefit from low error outliers. A less accurate model with fewer outliers, that shows the correct trends, is more useful than a model that is more accurate on average but has a large error for certain workloads.

7. RESULTS AND DISCUSSION

We apply the above modeling technique to the two processors considered. In both cases, we created models based on the full set of benchmarks, excluding the validation benchmarks, as well as based on the two reduced sets. The models were then validated using application benchmarks from the AMD APPSDK and Rodinia benchmark suite.

For the CPUs, we considered models ranging in complexity. The simplest models use a single predictor, such as the total number of dispatched instructions, while the most complex ones contain up to 25 terms and consider the instruction and memory types separately. We also considered a range of models for the GPU. The simplest model uses a single predictor, while the most complex uses 18.

Figure 5 show the training and validation error of the three benchmark sets for three representative models of the Xeon's power. Model 0 considers only the total number of dispatched instructions, model 1 considers accesses to various functional units as well as each level of cache, and model 2 is the most accurate model for the Xeon and considers only the functional units. As expected, we can clearly see that in all cases the training error is lowest when we use the largest available data set for modeling. In particular, while the training error for all models is below 6% with the full model, it can more than double when using the reduced models. However, we find that for our best model, this has a much smaller effect on the validation error. We observe mean validation errors of 2.5%, 3.4%, and 2.6% for the full, reduced, and minimal set of benchmarks respectively. The model produced using the minimal set is specified in Table 7.

There are two main reasons that we see a lower validation than training error for model 2. The first reason is that this model does not consider any memory related predictors beyond the calculation of effective addresses, therefore the cache micro-benchmarks have a higher error. For the minimal case, they increase the training error rate by 2.2%. The second reason is that the *canneal* benchmark has a very high error rate, 27% on average and up to 55% for the four core case.

Figure 6 shows the results for the Fusion's CPU using three models. The first two models use the same predictors as the Xeon's. Model 2 is again the best model, and this time it includes the functional units, as well as the number of DRAM accesses. On the Fusion, using reduced benchmark sets has even less of an effect on training error. In most cases, the error rate increases less than 1%. This shows that we can substantially reduce the number of benchmarks used and generate models of similar accuracy, if the benchmarks are

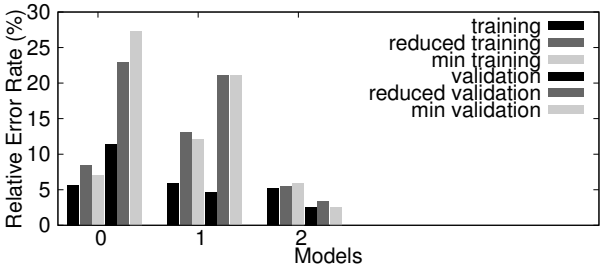

Figure 5: Model errors on the Xeon CPU.

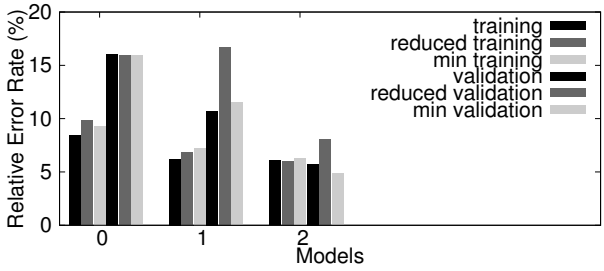

Figure 6: Model errors on the AMD Fusion CPU.

Table 6: Xeon Minimal Model

Predictor	Coefficient (J)
Integer multiplication	$7.74e^{-10}$
Integer divide	0 $6.23e^{-11}$
Effective address	$1.01e^{-9}$
Floating-point comparison	$4.50e^{-9}$
Floating-point complex	$5.38e^{-10}$
SSE complex	$5.32e^{-11}$
Static Power	
Global	12.7 W
per-core	4.5 W

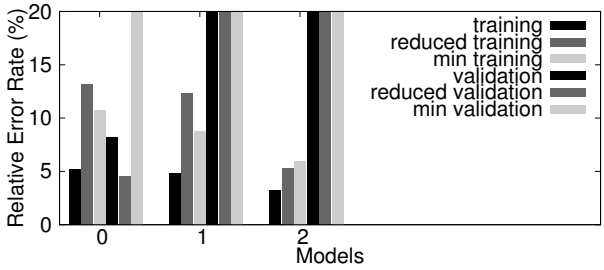

Figure 7: Model errors on the AMD Fusion GPU.

carefully chosen. We also see that the minimal set has lower validation error than the reduced set. By picking a single benchmark per cluster, we have a better distribution of the various energy consumptions. For the most accurate model, we observe mean validation errors are 5.7%, 8.1%, and 4.9% for the full, reduced, and minimal set of benchmarks, respectively. For model 2, *canneal* is again an outlier, with a mean error of 13% and a 27% error rate in the four core case. Since this model has a memory related predictor, the cache micro-benchmarks only increase the training error rate by 0.5%.

Figure 7 shows the results for the Fusion's GPU using three models. Model 0 considers the number of execution cycles, model 1 considers the cycles for the three functional units separately, and model 2 considers the number of instructions per functional unit, as well as the two types of memory accesses. We see that the more complex models overfit the data, which results in very poor predictive ability[1]. Only the first model is accurate and the validation errors are 8.2%, 4.5% and 110% for the full, reduced, and minimal set of benchmarks, respectively. The minimum set produces such poor results because the GPU benchmarks form few clusters, resulting in only 32 benchmarks being used in this case, giving a model with poor predictive ability. The fact that a simple model such as model 0 suffices to describe an integrated GPU has also been shown by Bircher and John [26].

The full model for the Fusion heterogeneous processor combines the best CPU and GPU models. We have to ensure that we are not double counting the static power consumption of shared resources. To find this static power, we measured the idle power consumption of the Fusion and found it to be 8.2 W. This value is then subtracted from the static power consumption of both the CPU and GPU, allowing for the situation when they are both active. We validate the final model using the Rodinia heterogeneous benchmarks.

[1]All the cutoff bars have error rates above 100%.

Figure 8 shows the total measured energy consumption of the benchmarks, and predicted values using both the full, and reduced models. The Reduced model is composed of the reduced GPU and minimal CPU models. The mean error across these benchmarks is 8.7% and 7% respectively, which means both methods produce very accurate models. Table 7 specifies the reduced APU model.

8. RELATED WORK

Most of the previous power modeling work has focused on CPUs. Only recently have these approaches been applied to GPU power modeling. Some works took advantage of existing CPU circuit-level modeling infrastructure and added it to the GPGPUSim simulator. Work by Wang et al. [27] added power modeling to GPGPUSim using modified versions of Wattch and Orion [28]. Both GPUWattch [29] and GPUSimPow [30] combine GPGPUSim with modified versions of McPAT to create GPU power models. These works model Nvidia hardware.

There has also been work to create statistical power models for GPUs. Hotpower [31] is an Nvidia GPU power model based on GPU PMCs, as is the model developed by Nagasaka et al. [32]. Wang et al. [33] developed a power model based on the PTX assembly code generated by the Nvidia kernel compiler. The model developed by Hong and Kim [34] is based on access rates obtained from their PTX assembly code analysis tool [35]. These top-down approaches generate models that can be useful for online power estimation in real hardware and can also be used to evaluate the effect of program code on power consumption, but they have limited use for architecture evaluation.

There has been comparatively little prior work looking at AMD GPUs or integrated GPUs. Only one model of an integrated GPU was presented by Bircher and John [26] and it was part of a top-down full system power model. The

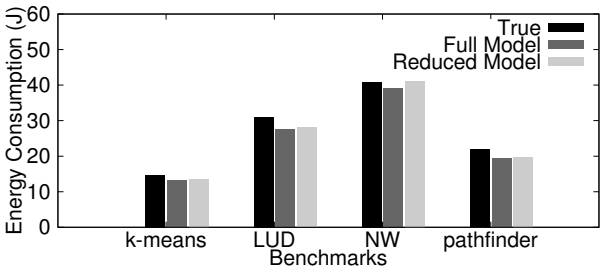

Figure 8: Validation of the full Fusion models.

Table 7: Reduced APU Model

Predictor	Coefficient (J)
Integer multiplication	$1.03e^{-09}$
Integer division	$9.95e^{-08}$
Bitwise	$1.57e^{-09}$
Floating-point multiplication	$8.12e^{-09}$
SSE bitwise	$7.29e^{-10}$
SSE floating-point comparison	$2.52e^{-09}$
DRAM access	$2.33e^{-07}$
Static Power	
Global	8.2 W
CPU	12.6 W
CPU Per-core	3.3 W
GPU	6.3 W
GPU Per-core	0.07 W

only model for a discrete AMD GPU was presented by Ma et al. [36] which used GPU performance counters, but measured power at the system level. Measuring power at the system level can be very inaccurate, as it assumes that all components other than the GPU have a constant power consumption, which is not the case. Both these approaches are also top-down and thus have the associated limitations.

There has been little work looking at instruction clustering based on energy consumption. The work by Bona et al. [37] aims to create an EPI power model. They compute k-means with five to sixteen clusters for a 60-instruction ISA. They show that the energy consumption of only 11 representative instruction can represent the ISA, while keeping the standard deviation per cluster below 13%.

There has also been some work by Wang et al. [38] to study power budgeting for heterogeneous processors. However, their power model is very coarse assuming constant per-core power consumption. This means that the only way to alter the CPU's or GPU's power consumption is to increase or decrease the core count. Neither the effect of the workload on power consumption, nor frequency scaling are considered.

9. CONCLUSION

We show that bottom-up power modeling techniques developed for CPUs can be effectively extended to heterogeneous APUs. In so doing, we create the first bottom-up power model of a heterogeneous processor. The decomposable nature of bottom-up models allows them be used for similar architectures with different core counts or operating frequencies.

We also demonstrate that it is possible to cluster microbenchmarks based on power consumption to reduce the number of benchmarks required to create accurate power models. By ensuring that a reduced benchmark set still covers a wide range of energy consumptions and operational variety, we were able to reduce the number of compute microbenchmarks used in modeling by up to 93%. This has comparatively little effect on the quality of the model, as illustrated by the fact that our final APU models have error rate of 8.7% and 7% for the full and reduced benchmark sets, respectively. This significantly reduces the effort required to create accurate power models, since only the power measurement, which run quickly on real hardware, needs to be made for all of the benchmarks. The much more time consuming simulations only need to be performed for a representative sub-set of the benchmarks, resulting in much lower computational requirements.

Acknowledgments

We thank the reviewers for their feedback, Steven Gurfinkel for the helpful discussions, and the members of Prof. Enright Jerger's research group for their suggestions. This work was supported by NSERC, OCE, and AMD. Computations were performed on the Gravity supercomputer at the SciNet HPC Consortium. SciNet is funded by: the Canada Foundation for Innovation under the auspices of Compute Canada, the Government of Ontario, Ontario Research Fund - Research Excellence, and the University of Toronto.

10. REFERENCES

[1] T. Mudge, "Power: A first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.

[2] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: a simulation framework for CPU-GPU computing," in *Proceedings of the international conference on Parallel architectures and compilation techniques*, 2012, pp. 335–344.

[3] D.-P. Pop, "A look at Intel's new Nehalem architecture: The Bloomfield and Lynnfield families and the new Turbo Boost technology," *Journal of Information Systems & Operations Management*, vol. 3, no. 2, pp. 410–416, December 2009.

[4] A. Branover, D. Foley, and M. Steinman, "AMD Fusion APU: Llano," 2012, pp. 28–37.

[5] AMD, *Software Optimization Guide for AMD Family 10h and 12h Processors*, 2010. [Online]. Available: http://developer.amd.com/resources/documentation-articles/developer-guides-manuals/

[6] ——, *Evergreen Family Instruction Set Architecture Instructions and Microcode*, 2011. [Online]. Available: http://developer.amd.com/wordpress/media/2012/10/AMD_Evergreen-Family_Instruction_Set_Architecture.pdf

[7] DATAQ Instruments, "DI-149 8-channel USB data acquisition starter kit," 2012.

[8] Intel, *ATX Specification*, 2003.

[9] Allegro MicroSystems, "Hall effect linear current sensor with overcurrent fault output for <100 v isolation applications," *ACS711 datasheet*, 2008.

[10] AMD, "Why is AMD CPU/APU not always running at full speed?" Aug. 2012. [Online]. Available: http://support.amd.com/us/kbarticles/Pages/CPUAPUnotrunningatfullspeed.aspx

[11] V. Pallipadi, "Enhanced Intel speedstep technology and demand-based switching on Linux," *Intel Developer Service*, 2009.

[12] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba, *Advanced Configuration and Power Interface Specification*, Dec. 2011. [Online]. Available: http://www.acpi.info/spec50.htm

[13] V. M. Weaver, "Linux perf event features and overhead," in

the FastPath Workshop, International Symposium on Performance Analysis of Systems and Software, 2013.

[14] AMD, "AMD accelerated parallel processing (APP) SDK," 2011. [Online]. Available: http://developer.amd.com/sdks/amdappsdk/pages/default.aspx

[15] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *Proceedings of the international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[16] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the International Symposium on Workload Characterization*, 2009, pp. 44–54.

[17] T. Diop, S. Gurfinkel, J. Anderson, and N. Enright Jerger, "DistCL: A framework for the distributed execution of OpenCL kernels," in *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2013.

[18] G. Hamerly and C. Elkan, "Learning the k in k-means," in *NIPS 16*, S. Thrun, L. Saul, and B. Schölkopf, Eds. Cambridge, MA: MIT Press, 2004.

[19] J. A. Hartigan, *Clustering algorithms*. John Wiley & Sons, Inc., 1975.

[20] T. W. Anderson and D. A. Darling, "Asymptotic theory of certain goodness of fit criteria based on stochastic processes," *The Annals of Mathematical Statistics*, vol. 23, no. 2, pp. 193–212, 1952.

[21] A. N. Pettitt, "Testing the normality of several independent samples using the Anderson-Darling statistic," *Journal of the Royal Statistical Society*, vol. 26, no. 2, pp. 156–161, 1977.

[22] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *SIGARCH Computer Architecture News*, vol. 28, no. 2, 2000, pp. 83–94.

[23] R. Bertran, M. Gonzàlez, X. Martorell, N. Navarro, and E. Ayguadé, "Counter-based power modeling methods: Top-down vs. bottom-up," *The Computer Journal*, vol. 56, no. 2, pp. 198–213, 2013.

[24] C. L. Lawson and R. J. Hanson, *Solving least squares problems*. SIAM, 1974, vol. 161.

[25] K. M. Mullen and I. H. M. van Stokkum, "The nnls package," 2007.

[26] W. Bircher and L. John, "Complete system power estimation using processor performance events," *Computers, Transactions on*, vol. 61, no. 4, pp. 563–577, April 2012.

[27] G. Wang, "Power analysis and optimizations for GPU architecture using a power simulator," in *Proceedings of the International Conference on Advanced Computer Theory*, vol. 1, 2010, pp. V1–619.

[28] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: a power-performance simulator for interconnection networks," in *Proceedings of the International Symposium on Microarchitecture*, 2002, pp. 294–305.

[29] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling energy optimizations in GPGPUs," in *Proceedings of the International Symposium on Computer Architecture*, 2013.

[30] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "How a single chip causes massive power bills GPUSimPow: A GPGPU power simulator," in *Proceedings of the Symposium on Performance Analysis of Systems and Software*, 2013, pp. 97–106.

[31] X. Ma, M. Dong, L. Zhong, and Z. Deng, "Statistical power consumption analysis and modeling for GPU-based computing," in *Proceedings of the SOSP Workshop on Power Aware Computing and Systems (HotPower)*, 2009.

[32] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Proceedings of the International Green Computing Conference*, 2010, pp. 115–122.

[33] H. Wang and Q. Chen, "Power estimating model and analysis of general programming on GPU," *Journal of Software*, vol. 7, no. 5, pp. 1164–1170, 2012.

[34] S. Hong and H. Kim, "An integrated GPU power and performance model," in *SIGARCH Computer Architecture News*, vol. 38, no. 3, 2010, pp. 280–289.

[35] ——, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *SIGARCH Computer Architecture News*, vol. 37, no. 3, pp. 152–163, 2009.

[36] Y. Zhang, Y. Hu, B. Li, and L. Peng, "Performance and power analysis of ATI GPU: A statistical approach," in *Proceedings of the International Conference on Networking, Architecture, and Storage*, 2011, pp. 149–158.

[37] 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," in *Proceedings of the Design Automation Coference*, 2002, pp. 886–891.

[38] H. Wang, V. Sathish, R. Singh, M. J. Schulte, and N. S. Kim, "Workload and power budget partitioning for single-chip heterogeneous processors," in *Proceedings of the international conference on Parallel architectures and compilation techniques*, 2012, pp. 401–410.