Title
Equation-Based Power Model Integration in ESESC

Permalink
https://escholarship.org/uc/item/88q24991

Author
Sinha, Meeta

Publication Date
2013

Peer reviewed|Thesis/dissertation
EQUATION-BASED POWER MODEL INTEGRATION IN ESESC

A thesis submitted in partial satisfaction of the requirements for the degree of

MASTER of SCIENCE

in

COMPUTER ENGINEERING

by

Meeta Sinha

June 2013

The Thesis of Meeta Sinha is approved:

__________________________
Professor Jose Renau, Chair

__________________________
Professor Andrea Di Blas

__________________________
Professor Anujan Varma

Tyrus Miller
Vice Provost and Dean of Graduate Studies
# Table of Contents

List of Figures v  
List of Tables vi  
Abstract vii  
Acknowledgments ix  
1 Introduction 1  
2 Related Work 7  
  2.1 ESESC 7  
  2.2 Tools for Power Modeling 9  
    2.2.1 McPAT 10  
    2.2.2 CACTI 12  
    2.2.3 Wattch 17  
  2.3 Analytical Models 19  
3 Power Model 23  
  3.1 Power Equation 25  
    3.1.1 Modeling Technique 25  
    3.1.2 SRAM and Cache Power Model 29  
  3.2 Power Model Integration 31  
    3.2.1 Existing ESESC Power Model 31  
    3.2.2 LibPeq : New ESESC Power Model 35  
4 Performance Evaluation 41  
  4.1 Speed Gain 42  
  4.2 Comparison With Original ESESC 46  
5 Conclusion 49  
  5.1 Future Work 50
# List of Figures

2.1 Block Diagram of McPAT framework ........................................ 10  
2.2 Logical Organization of a cache ...................................... 15  
2.3 Block diagram of the Wattch framework .............................. 18  

3.1 (a) old approach for power computation in ESESC (b) new approach for  
    power computation in ESESC ........................................ 24  
3.2 Pareto Optimality ......................................................... 28  
3.3 Hierarchical energy model used in ESESC ......................... 32  
3.4 Ten-stage pipeline model for OoO processor .................... 33  

4.1 Initialization time ....................................................... 42  
4.2 Real time for two billion instructions .............................. 44  
4.3 Real time for 15 billion instructions ............................... 44  
4.4 Renaming unit ............................................................... 46  
4.5 Instruction Fetch Unit .................................................. 47  
4.6 Register Files ............................................................... 47  
4.7 Reorder Buffer .............................................................. 48  
4.8 Icache ................................................................. 48  

List of Tables

3.1 Partitioning criteria for the power model equation . . . . . . . . . . . . 29
3.2 Coefficients for SRAM and cache energy equations . . . . . . . . . . . . 30
Abstract

Equation-Based Power Model Integration in ESESC

by

Meeta Sinha

Due to increasing complexity of microprocessor design and a lot of stress on high performance, power consumption and associated heat have reached critical levels. Therefore, the simulation methodology today is adding a new dimension: power optimization. Modern computer architects need to do a comprehensive analysis of power/performance tradeoffs to do a fair design evaluation. Array-based memory structures account for the largest share of power consumption in modern processors. Also, as these components have regular structure, it is easier to model them and quantify their power consumption. As a result, power-reduction approaches primarily focus on memory structures. For array-based components, such as SRAM, cache, CAM, the design search space is very wide. There are large numbers of parameters that need to be optimized given certain main parameters such as size, number of ports, associativity, bus width, etc. The simulation platform must iterate through these parameter values to find the optimal configuration for a given design constraint. The problem is that iterating through these many parameters is extremely slow (may require days) and resource-intensive.

There are energy models such as CACTI, Wattch, and McPAT that can be integrated with architectural simulators to estimate power consumption. However, the issue of fast and comprehensive power consumption analysis has not been addressed well. In
our thesis, we focus on improving the time taken by ESESC (Enhanced-SuperESCalar simulator) to estimate power consumption for SRAM and cache-based structures. The new equation-based power model that we have integrated in ESESC was developed by doing a full design space exploration using CACTI. Using this new power model, called LibPeq, we get an average speed gain of 16.8% for large simulation runs and an impressive 99.98% improvement in initialization time. The new power model can also be integrated with other simulators and energy models that rely on CACTI for power estimation of array-based structures. ESESC, along with this new power model, can be a very strong tool for modern computer architects and researchers looking for power-efficient designs.
Acknowledgments

I would like to express my sincere gratitude towards my guide Prof. Jose Renau for his constant support, vision, and encouragement. He is a wonderful advisor who is truly committed towards every project he is guiding. His depth of knowledge about the subject was crucial in overcoming hurdles. One simply could not wish for a better or friendlier supervisor. I will be always indebted to him for giving me the opportunity to work under him on this thesis.

I would also like to thank my committee members Prof. Andrea Di Blas and Prof. Anujan Varma for their support, patience, and insightful comments.

The guidance and help received from other students in MASC lab was vital for the successful completion of this project. Special thanks to Ehsan Ardestani for his time, support, and ideas. His thorough understanding of ESESC and his patience in explaining things are commendable. I would also like to thank Elnaz Ebrahimi for her encouraging words and support.

No task can ever be completed successfully without parents blessings; I would like to express my deepest gratitude to my amazing parents, Pratima and Ramesh, for giving me the encouragement to pursue my dreams. Finally, I would like to thank my wonderful husband Dr. Ashutosh Verma. He was always there cheering me up and stood by me through good and bad times. His love, patience, and encouragement allowed me to embark on this journey and to finish it.
Chapter 1

Introduction

Power dissipation and consequent heat issues have become a critical bottleneck in processor performance improvement. Modern architectures deploy multiple cores, multiple threads, and large multiple-level caches to get higher speed and better overall performance. Such high-performance modern processors have high power requirement as well. The direct correlation between power consumption and thermal dissipation implies that the higher the power consumption the higher the temperature. Existing cooling techniques (e.g. heat-sink, fan) are failing to keep up with the amount of heat generated in modern microprocessors. Designing microprocessors that perform tasks faster and more efficiently without overheating is a major consideration of nearly all computer architects. As a result, power efficient architecture is a very active research area and there is a great demand for fast and accurate power modeling and simulator tools.

The idea behind developing a microprocessor architecture simulator is to en-
able a microprocessor researcher to evaluate his/her design without going through the expensive process of actual fabrication. The most desired features of any simulator are accuracy and speed. Multiprocessor simulation is approximately one million times slower than real processor [34]. Speed is of even more essence for university research groups working on tight computing budgets and deadlines. There are different types of simulators available, e.g. some simulators use instruction traces of applications while other actually execute the application. Basically, the simulators run benchmarks which are designed to mimic a certain type of workload for a particular hardware structure or system. Many academic simulators today also support power modeling extension to help architects do a detailed performance/power tradeoff analysis. Such simulators, like ESESC (Enhanced Super ESCalar Simulator) have two main components: the emulator, which executes the benchmark, and a timing simulator, which models the timing and power consumption. Modern simulators designed for today’s complex microprocessor architectures are extremely sophisticated software systems. ESESC written in C++ and developed at UCSC (University of California, Santa Cruz), is designed to be a very fast simulator that can model superscalar Out-of-Order processors with heterogeneous multicores. ESESC enables a designer to do detailed analysis of performance, power consumption and heat dissipation. ESESC was developed to be an enhanced version of one of the most attractive academic simulators: SESC (Super ESCalar Simulator). Existing version of ESESC uses McPAT (Multicore Power Area and Timing) [13] and CACTI [14] for calculating power dissipation. CACTI calls for computing power consumption of array-based memory structures is the main cause of slowdown. In our thesis,
the main focus is to provide a framework for integrating an equation-based power model to ESESC, in order to speed up power consumption analysis. The accuracy analysis of the equations used in our model is present in Ebrahimi MS thesis [10] and is out of scope for our work.

McPAT provides an integrated power, area, and timing modeling framework. It is a very good power modeling tool with some excellent features that makes it stand out from other similar tools. First, McPAT specifies the low-level design parameters of regular components (e.g. interconnects, caches, and other array-based structures) based on high-level constraints (clock rate and optimization target) given by a user. This allows the user to have the option of ignoring low-level details of the circuit being modeled. Second, as static power has become comparable to dynamic power, it is important to take into account leakage and short-circuit power along with dynamic power. McPAT allows us to take into account all forms of power dissipation. Third, McPAT can model power for most of the important parts of multicore processors such as caches, interconnects, memory controllers, and clocking circuit. Therefore, McPAT provides an excellent framework for performing power-consumption analysis of multicore/manycore and multithreaded complex processors. McPAT relies on CACTI to compute power consumption for array-based memory structures. The array-based structures contribute most to power consumption in any microprocessor [5, 29, 18, 17]. Hence, speeding up this aspect of power computation can have a major impact on overall performance of ESESC.

CACTI is one of the oldest and most popular tools to calculate power, area,
and timing estimates for processors. Its main focus is on array-based structures such as SRAM and caches. CACTI contains optimization features that allow it to find a configuration with minimal power consumption, given constraints on area and timing. It has evolved over time to include newer technologies and more parameters in order to improve its accuracy. For instance, its newer version uses models based on ITRS (International Technology Roadmap for Semiconductors) instead of linear scaling, and its latest version is capable of modeling non-uniform cache accesses and different types of wires such as RC-based wires with different power, delay, and area characteristic. CACTI as a tool has widely been used by computer architects either directly for modeling SRAM-based caches and plain memories, or indirectly through other tools such as McPAT and Wattch.

The input for CACTI consists of parameters such as cache capacity, block size, associativity, technology generation, number of ports, and number of independent banks. As output, it produces the cache configuration that minimizes delay, along with its power and area characteristics. The CACTI search space can be expanded if more parameters are specified before each run. In order to observe how power, energy, or delay values vary for each particular set of main parameters such as technology generation, size, and number of ports, one would have to perform a number of simulations, varying all parameters, which can reach up to several thousands. Covering the whole design search space requires many hours of computation, and therefore only a subset is covered by default in CACTI. It is desirable to do a full design sweep of all possible parameters before making any design decision for SRAM or cache. The equation-based energy
model used in our work was developed by running thousands of CACTI simulations so that it covers the full design space exploration.

The new energy model has been extracted from the Energy-Delay-Product ($EDP$) equation and delay equation given in Ebrahimi MS thesis [10]. $EDP$ metric was initially proposed by Gonzalez and Horowitz [15] to evaluate tradeoffs between circuit-level power-saving techniques and performance. In general, processors trade energy for performance as it is quite challenging to improve both performance and energy simultaneously. In order to improve $EDP$ of a microprocessor, one needs to either increase the performance or reduce the power dissipation without adversely affecting the other quantity. $EDP$ is a great optimization metric because it allows for system architecture and block-level optimization, and it guides the optimization to yield a globally-optimal solution. The $EDP$ equation and delay equation was developed using Pareto optimality followed by non-linear regression and optimization on results obtained from thousands of CACTI simulation runs for covering all design parameters in SRAM and cache design space. Pareto efficiency approach allows the designer to make tradeoffs between parameters such as energy and delay.

In a nutshell, to address all the above-mentioned issues, our main goal is to speed up the ESESC simulation by replacing CACTI with a new power model tool called LibPeq (Library Power Equation model). LibPeq uses dynamic energy equations for SRAM and cache derived from $EDP$, and delay equation given in Ebrahimi MS thesis [10]. In the new ESESC, McPAT uses LibPeq instead of CACTI to compute power for array-based structures SRAM and cache.
The rest of the thesis is organized as follows. In Chapter 2 we describe some interesting concepts and power-modeling tools related to our work. First, we talk about the ESESC simulator and the sampling methodology that makes it stand out from other simulators for modern multicore/multithreaded processors. Then, we give a quick review of different energy-aware simulators, namely McPAT, CACTI, and Wattch. Next, we discuss about different analytical models for computing power consumption. Chapter 3 provides an in-depth overview of how the \textit{EDP} and delay equations used for deriving our energy model were developed, the details of the current approach for computing power consumption in ESESC, and a synopsis of LibPeq integration to ESESC. Chapter 4 gives performance evaluation of our new enhanced ESESC with LibPeq. Chapter 5 concludes our thesis and includes a summary of our work and directions for future developments.
Chapter 2

Related Work

This section first gives a quick overview of the sampling technique that makes ESESC truly stand out from other similar simulators designed for power consumption/thermal analysis. Then, it describes McPAT, CACTI, and Wattch frameworks. Since our work is mainly focused on SRAM and caches, we provide details of internal functioning of a cache in our CACTI section. Finally, we discuss some related past research dealing with analytical model generation.

2.1 ESESC

Simulation is the main approach used by researchers to evaluate the performance of any new computer architecture design. Unfortunately, running benchmarks on a simulator is on the order of million times slower than actual hardware. Moreover, simulating multithreaded applications further aggravates this problem as most simulation tools are single-threaded. Sampling the benchmark instructions is a common approach
for speeding up the simulation process and the sampling technique used is crucial for getting unbiased estimates [33]. The main goal of any sampling approach is to estimate parameters of benchmark programs on a simulated microarchitecture and have good fidelity and speed. A lot of research has been done to improve sampling techniques and some popular statistical sampling approaches are SMART (Sampling Microarchitecture Simulation), TurboSMARTS, SMARTSim, and SimFlex [33, 34, 35].

The current trend in computer architecture is towards multicore and multithreaded processors [30, 24]. There have been many attempts towards developing multiprocessor architecture simulators that are themselves multithreaded, but the speedup gained has not been very satisfying. Commonly used simulators simulate a multithreaded program sequentially [11, 9, 12, 23]. Sampling in multicore systems is more challenging because the standard metric for performance evaluation, IPC (Instructions Per Cycle) does not truly reflect the performance of multithreaded workloads running on multiprocessors [1]. It has been shown that the relative error in IPC speedup generally increases as the number of processors increases. Sampling could also distort the overlap of threads and lead to inconsistency in progress among threads. It has been shown that for multithreaded configurations with programs having significant sharing and contention patterns, it is necessary to define the sampling parameters with respect to progress in time rather than to the number of instructions, to avoid divergence of progress among threads and to allow fair temperature simulation of the design [4].

Basically, there are no sampling models for multithreaded application simulations or multicore temperature simulations. ESESC was developed to bridge this gap,
and it uses an innovative sampling technique called TBS (Time Based Sampling) [4]. ESESC is fast (simulations speed of 9 MIPS) and provides a holistic set of tools for performance evaluation of new innovative computer architecture designs. TBS provides a fast sampling framework to evaluate the performance, power, and temperature trade-offs in any multicore configuration running multithreaded program. TBS is the only sampling approach that enables thermal evaluation of shared memory systems running multithreaded applications.

Traditional sampling techniques specify the sampling parameters such as sample length or interval between samples as a function of number of instructions. On the other hand, in TBS the sampling sub-intervals (i.e. memory system warmup, detailed warmup, and detailed timing) are defined to have a fixed number of cycles. TBS adjusts the number of instructions at runtime to keep length and duration of each sub-interval constant. Therefore, the length is converted at runtime from number of cycles to number of instructions based on observed performance, and this is done independently for each thread. The evaluation of TBS shows an average error of 5.5% and 2.4% for power and maximum temperature respectively.

2.2 Tools for Power Modeling

There is a multitude of tools available for modeling power. The most prominent ones are McPAT, CACTI, and Wattch.
2.2.1 McPAT

McPAT (Multicore Power, Area and Timing) is an integrated power, area, and timing modeling framework that enables architects to perform comprehensive design space exploration [13]. It uses new metrics such as Energy-Delay-Area Product (EDAP), and EDP that combine performance with both power and area. Such metrics are more effective than traditional metrics such as power and energy for quantifying the cost of new architectural ideas. Power, area, and timing need to be studied together more accurately as technology keeps scaling down. Figure 2.1 gives an overview of the McPAT framework.

McPAT has a XML interface that allows it to interface and integrate with multiple performance simulators. The key components of McPAT are:

1. the hierarchical power, area, and timing models;

2. the optimizer for determining circuit level implementations,
3. the internal chip representation that drives the analysis of power, area, and timing.

Most of the parameters in the internal chip representation, such as cache capacity and core issue width, are directly set by the input parameters.

McPAT’s optimizer focuses on two major structures, namely arrays and interconnects. The optimizer determines missing parameters in the internal chip representation. It generates the final chip presentation which is used to compute the final area, timing, and peak power. The peak power of individual units and the component utilization statistics (activity factor) is used to compute the final runtime power dissipation. The activity factor of a component is the average number of times its output changes value per number of clock periods. McPAT is capable of computing all three forms of power consumption: dynamic, leakage, and short-circuit. Equation 2.1 describes the formula for computing dynamic power.

\[
P_d = C V_{dd}^2 \cdot a \cdot f
\]

\(C\) is the load capacitance and depends on circuit and transistor size, \(V_{dd}\) is the supply voltage and depends on process technology, \(f\) is the clock frequency, and \(a\) is the activity factor and it indicates how often clock transitions lead to actual switching activity, on average. Switching circuits dissipate short circuit power due to momentary short through pull-up and pull-down devices. Leakage power is attributed to the transistors because a small current passes through the source and drain of off-state transistor. Equation 2.2 shows the leakage current \(I_{ds}\):  

\[
I_{ds} = I_{ds0} \exp\left(\frac{V_{gs} - V_t}{n \cdot vT}\right) \left(1 - \exp\left(-\frac{V_{ds}}{vT}\right)\right)
\]
where, $V_{gs}$ is the gate-source voltage, $V_{ds}$ is the drain-source voltage, $v_T$ is the thermal voltage, $V_t$ is the threshold voltage, $I_{ds0}$ is the leakage current when $V_{gs} = V_t$ and $V_{ds} = 0$, and $n$ is a constant coefficient for a given technology. Leakage power is obtained by multiplying leakage current with supply voltage and dividing the product by total time. Another factor contributing to leakage power are the gates, because some current leaks through the gates.

The main focus of McPAT is to accurately model power and area for a given target clock rate. The designer specifies the target clock frequency, the area and power deviation, the optimization function, and other architectural circuit and technology parameters. The bottleneck is that the optimization space that McPAT explores can be huge, especially when there are many unspecified parameters. McPAT uses some optimization function to report the final power and area values. Also, McPAT depends on CACTI for modeling array-based memory structures.

### 2.2.2 CACTI

CACTI was designed to model cache and memory access time, cycle time, area, leakage and dynamic power [14]. It was designed in 1994 to help computer architects better understand the performance tradeoffs inherent in memory system organization. It is widely used and has undergone a lot of fine tuning in order to keep pace with changing processor trends. New features were added to improve its applicability and accuracy. Modern processor design trend is inclined towards large on-chip caches. The properties of large caches depend a lot on the characteristic of the interconnection networks that
connect various sub-modules of a cache. Hence, the latest version of CACTI has been designed to enable it to model large caches. It can model Non-Uniform Cache Access (NUCA) and different types of routers and wires, such as RC-based wires with different power, area, and delay characteristics. The latest version is able to identify cache configurations that can reduce power by a factor of three while incurring only a 25% delay penalty [26].

CACTI can take cache parameters such as cache size, block size, associativity, and technology as command-line arguments, or the user can specify them in a configuration file that allows users to describe the parameters in much greater detail. Some of the important parameters that are included in the configuration files are: number of read, write, and read-write ports, operating temperature (used for computing leakage power), cache type (DRAM, SRAM or a simple RAM such as registers that don’t need tag array), number of cores, cache level L2/L3, NUCA bank account, and cache access mode. CACTI supports four different process technologies (90 nm, 65 nm, 45 nm, and 32 nm) with process-specific values obtained from ITRS.

Shown in figure 2.2 is a description of the logical organization of a cache [14]. In a cache, the decoder takes the requested main memory address as input. The address decoded then activates a word-line in data array and tag array. Each row comprises of memory cells and each memory cell is connected with a pair of bit-lines. The contents of an entire row are placed on the corresponding bit-lines, which are then sensed using sense amps. There is a multiplexer before the sense amplifier because multiple bit-lines share the same sense amplifier. Next, the data from the tag array is compared with
the tag bits of the input address. This comparator logic output drives the multiplexer that selects the appropriate data from the data array and sends it to the requesting processor. In addition to the ones mentioned earlier, the cache access model of CACTI takes following parameters as input:

- the cache size,
- the block size/line size,
- the associativity,
- the number of ports,
- the number of independent banks and
- the technology generation.

The delay, power, and area are computed for the following components:

- decoder,
- word-lines,
- bit-lines,
- sense amplifiers,
- comparators,
- multiplexer drivers,
- output drivers, and
Figure 2.2: Logical Organization of a cache

- inter-bank wires.

The delay is calculated for both data and tag array. Each component is decomposed into several equivalent RC circuits, and the resistance and capacitance of each component is calculated. CACTI partitions each storage array (in both horizontal and vertical dimensions) to produce smaller sub-arrays by defining three parameters:

- $Ndwl$: number of segments per word-line of data array,
- $Ndbl$: number of segments per bit-line of bit-array, and
- $Nspd$: number of sets mapped to a wordline.

The word-line and bit-line delays are quadratic functions of the horizontal and vertical
dimensions of the array. Therefore, by dividing an array into sub-arrays reduces word-line and bit-line delays. Each sub-array has its own decoder, and a central pre-decoding is done to map the request to the correct sub-array. CACTI does an extensive search across different sub-array counts and sub-array aspect ratios to compute cache organization with optimal total delay. After calculating the delays, the total access time and cycle time are computed. As output, CACTI produces the cache configuration that minimizes delay, along with its power and area characteristics.

Equation 2.3 describes the default cost function used in CACTI in order to evaluate performance of a cache organization. The parameters taken into account are access time (acc_time), dynamic power (dyn_power), leakage power (leak_power), cycle time (cycle_time), and area. The user assigns weights for each of these terms in the configuration file.

\[
\text{cost} = W_{\text{acc\_time}} \frac{\text{acc\_time}}{\text{min\_acc\_time}} + W_{\text{dyn\_power}} \frac{\text{dyn\_power}}{\text{min\_dyn\_power}} + W_{\text{leak\_power}} \frac{\text{leak\_power}}{\text{min\_leak\_power}} + W_{\text{cycle\_time}} \frac{\text{cycle\_time}}{\text{min\_cycle\_time}} + W_{\text{area}} \frac{\text{area}}{\text{min\_area}}
\] (2.3)

Here $W_X$ and $\text{min}_X$ are the assigned weights and minimum value for any given parameter $X$, such as access time, and dynamic power.

CACTI also takes energy-delay ($ED$) or energy-delay-squared ($ED^2$) product value as input to its cost function to figure out a cache organization best for $ED$ or $ED^2$ product.

The CACTI search space can be expanded if more parameters are manipu-
lated before each run. Covering the whole design search space requires many hours of computation, and therefore, only a subset is covered by default in CACTI.

2.2.3 Wattch

Wattch is yet another power modeling tool, developed in 2000 [8]. It was designed to address two major drawbacks of the power analysis tools of the time, such as Powermill [16] and Quickpower:

- Most power analysis tools were designed to be used at a very late stage (i.e. after the layout was ready) of the design process.
- The simulation speed for analyzing multiple hardware configurations was prohibitively slow.

Wattch power modeling methodology is similar to that of McPAT and CACTI, and involves having parametrized power models of all major hardware structures of a microprocessor. Just like McPAT and CACTI, we can integrate Wattch with a wide variety of architectural simulator. But unlike McPAT and CACTI, Wattch does not model area and timing. Figure 2.3 illustrates the structure of Wattch:

All the processor components considered by Wattch model can be grouped into four categories:

- array-based structures, such as caches, register files, and branch predictors;
- fully-associative CAMs (content-addressable memory), such as TLBs;
Figure 2.3: Block diagram of the Wattch framework

- combinational logic and wires, such as functional units and result buses;
- clocking circuitry such as clock buffers and clock wires.

Dynamic power is a major portion of total power consumed and it is a function of the total load capacitance, the supply voltage, the voltage swing during switching, the clock frequency, and the activity factor. Equation 2.1 described the formula for dynamic power. The power consumed by a component is computed by partitioning the unit into stages, and by computing the capacitance of each stage. The physical implementation of array-based structures is estimated using CACTI. The array structures parameters are: number of rows, number of columns, and number of ports. These parameters are important because they affect the size and number of decoders, the number of word-lines, and the number of bit-lines which consume the most power in array structures. CAM structures are very similar to array structures, with the only difference being that they have tag-lines and match-lines instead of word-lines and bit-lines. For combinational logic and wires, the two main logic blocks considered are the instruction selection logic.
and the dependency check logic. The clocking network consumes a lot of power in high-frequency microprocessors. The main components of clock networks modeled for analyzing power consumption are: global clock metal lines, global clock buffers, and clock loading. The metrics supported by Wattch are: power (average and maximum power/cycle), performance (number of cycles for program execution), energy (product of power dissipation and execution time), and EDP. Compared to McPAT, Wattch uses older versions of energy models, like CACTI.

### 2.3 Analytical Models

Researchers have started focusing on developing analytical models using statistical techniques such as regression analysis and correlation to handle ever-expanding design space. Such models offer a computationally efficient and fairly accurate approach for predicting performance and power consumption of processor designs.

Lee and Brooks proposed a regression-based model for estimating microprocessor design efficiency and power consumption [19, 20, 21]. In regression modeling, domain-specific knowledge is used to specify predictors of a response. Some initial data is used to train and formulate the model. The correlation from initial data is then used for prediction. Overall, the model is developed from observed data by solving a system of linear equation, and the response is predicted by evaluating a linear equation which is easily done using well-developed linear algebra libraries. In their work, from a design space of approximately 22 billion possible configurations, 4000 sample observation con-
figurations/points were selected via simulations. These points were then used for doing regression analysis and generating models. For simulation, the authors selected twelve groups of parameters (e.g. physical registers, reservation stations, I-L1 cache, D-L1 cache, main memory etc.) that were varied simultaneously. Parameters within a group (e.g. for L2 cache the parameters were L2 cache size and L2 cache latency) were varied together to circumvent any fundamental design imbalances. The number of simulations required for considering all possible configurations of those parameters is more than 20 billion. Sampling configurations uniformly from these 22 billion choices is critical for the accuracy of the model being generated. Uniformly at random (UAR) technique is used to do the sampling. The main advantage of using random sampling is that it allows observation from the entire range of parameter values and that it enables identification of trade-offs and correlation between the parameter sets. UAR sampling ensures that the observations are not biased towards any particular configuration. Each randomly chosen configuration is evaluated using a randomly chosen benchmark. This work was unique in its approach because it used both architectural, and application-specific parameters as predictors for deriving power and performance models. The application specific predictors were taken from application characteristics such as branch stalls, branch mispredictions, and L1 misses when executing on a baseline configuration. Application baseline performance is the most significant predictor for its performance on other configurations. Significant predictors for the initial performance model were selected using data mining and machine learning approaches such as: variable clustering, association testing, correlation analysis of response-predictor relationship, and signifi-
cance testing using F-tests. The authors also suggest model optimization techniques to improve the accuracy of the response. For optimization, they propose the use of application-specific model for performance prediction and regional model (formulated using samples similar to predictive query) for power prediction. The main objective behind this work is to give a comprehensive analysis of microprocessor design space and to suggest how regression-based models can be derived and validated for their accuracy. The authors do not provide any model that can be used directly by architectures to speed up simulation.

Amrutur and Horowitz propose an analytical design model to compute area, delay, and energy consumption for SRAM [2]. The authors primarily take the delay model proposed by Wada et al. [32] and extend it to incorporate area and energy models. Their model is developed using certain assumptions to simplify the analysis. The authors render a fairly good analysis of how capacity and process technology impacts delay. The model provides a quantitative relationship between different components of SRAM delay and the size. This model allows the use of different technology nodes, and the optimization of parameters such as energy and delay. Like CACTI, the Amrutur and Horowitz model also does search across many configurations, but not across all possible parameters. The main issue in doing so is the amount of time it takes to iterate over all the parameters.

Ebrahimi MS thesis introduces a Pareto-optimal SRAM and cache models for predicting Energy Delay Product and Delay [10]. This model was developed by doing an exhaustive search of the entire CACTI 6.5 design space. The energy-delay
product prediction for cache has less than 2% average difference with the original CACTI result, and less than 9% difference for worst case cache configuration. For SRAM, the energy-delay product predictions are within 3% difference, and have a maximum of 15% difference for the worst configuration across all technologies. In our work we have used this equation as the base for deriving power equation for SRAM and cache-based architectural components. The details of how the equations were derived are given in the next section.
Chapter 3

Power Model

The primary focus of this thesis is to create a framework to incorporate equation based power model in ESESC. ESESC is an advanced version of one of the most popular academic simulator, SESC, and uses an innovative sampling technique (Time Based Sampling) which provides a very fast and accurate sampling framework to evaluate the performance, power and temperature trade-offs in any multicore configuration running multithreaded application. ESESC is the only simulator that has a sampling model for multithreaded application simulation or multicore temperature simulation. Increasing complexity of microprocessors introduces more design parameters and exponentially increases design space size. Performing thousands of simulations to do an exhaustive search of large microarchitectural design space increases simulation cost. To address this fundamental challenge, recent research is now focusing on using simulated results more efficiently via regression modeling and inference. Ebrahimi MS thesis provides regression based $EDP$ and delay equation for CACTI design search space. Prior
research work suggested statistical modeling technique for developing regression based model, but did not provide any physical model for CACTI design space. Since, ESESC calls to CACTI to compute power for array-based structures is a performance bottleneck, we replace CACTI with analytical energy model which has the added advantage of considering full sweep of CACTI design space. Figure 3.1 shows a comparison between the old approach and the new approach for power computation of array based structures in ESESC. This chapter discusses equation modeling technique and the power model integration approach.

Figure 3.1: (a) old approach for power computation in ESESC (b) new approach for power computation in ESESC
3.1 Power Equation

This section first gives a quick overview of the modeling technique used for developing $EDP$ and delay equation, as they are the base equations for LibPeq power model. Then, it describes details of the power equation and how it was derived from the $EDP$ and delay equation.

3.1.1 Modeling Technique

In this section, we discuss how the energy-delay product and delay equations were developed. Ebrahimi MS thesis generates Pareto-optimal equation using CACTI results, for estimating optimal energy-delay product and delay for SRAM and cache-based structures [10].

CACTI takes structure parameters as input and provides estimates about cycle time, access time, area, leakage, and dynamic power. For instance, a user can specify design objectives such as optimal access time, power or optimal energy delay product and input parameters such as cache size, associativity, number of ports, technology, I/O bus width, and number of banks in a configuration file. Using these input parameters, CACTI explores the design search space and gives suggestions for best possible cache structure configuration to achieve the design objective specified by user.

Equation 3.1 is $EDP$ or delay equation for SRAM and equation 3.2 is $EDP$ or delay equation for cache [10]. The log in the equation is natural logarithm, $X$ can be $EDP$ or delay, $a$ to $g$ are coefficients. $n$, $p$, $s$, $w$ and $l$ represent main parameters for
SRAM and cache.

\[
\log(X) = a + b \cdot \log(n) + c \cdot \log(p) + d \cdot \sqrt{s} + e \cdot \log(s) + f \cdot w + g \cdot \sqrt{w} \\
\text{(3.1)}
\]

\[
\log(X) = a + b \cdot \log(n) + c \cdot \sqrt{n} + d \cdot \log(w) + e \cdot (w/l) + f \cdot \log(s) + g \cdot \sqrt{s/w + l} \\
\text{(3.2)}
\]

The first step in deriving equation 3.1 and 3.2 involve analyzing the correlation between different SRAM and cache parameters, and their effect on output. Based on this analysis, four main parameters are selected, and the other remaining parameters are called optimization parameters. The main parameters for SRAM are: technology \((n)\), number of read/write ports \((p)\), SRAM size \((s)\) and bus width \((w)\). The main parameters for cache are: technology \((n)\), cache size \((s)\), cache associativity \((w)\), and cache line size \((l)\). The optimization parameters are the ones for which CACTI has to find the optimal value. Performing a full sweep of all possible values in optimization parameter search space may take days. For instance, full design sweep for a cache size of 4 KB, in 32 nm technology, and one-way associativity takes approximately 3 hours, while for a 1 MB cache with the same value of the other main parameters it takes 4-10 days, depending on machine being used. The 4 KB simulation was run on a quad-core AMD Opteron processor which runs at 2613 MHz and has a cache size of 1 KB. The 1 MB cache simulation was run on AMD Opteron Processor 6172 with 48 cores, 512 KB cache and 800 MHz speed.
One of the main objectives behind developing the analytical equations 3.1 and 3.2 was to expand the default CACTI search space. This was done by generating the analytical equation from a large database created by running thousands of CACTI simulations. For instance, a 32 nm, 64 KB, 2-way, 16 Byte-line cache configurations was run multiple times in order to cover all possible optimization parameter values. Approximately 132,000 CACTI simulations were done for SRAMs and 17,000 simulations for cache. R scripts and shell scripts were used to automate the simulation run for different values of main and optimization parameters. The result of each run was analyzed, and the dynamic energy and delay values were extracted. Then, Pareto analysis of these values was done.

Minimizing the $EDP$ value is the main optimization goal for this approach. Horowitz and Gonzalez introduced a new metric, $EDP$, for processor performance evaluation [15]. They show that compared to traditional metrics such as power and energy, $EDP$ is a better metric for modern microprocessor designs that deal with both performance and power optimization challenges. Power alone is not a good metric, as power is related to frequency and we can simply reduce clock speed to reduce power. This approach reduces power consumption but deteriorates processor performance. Similarly, using energy or Jules per Instruction as metric, has some drawback. Energy is a function of $CV^2$ ($C$ is capacitance and $V$ is voltage). Hence, reducing voltage or reducing capacitance (by using smaller transistors) can reduce energy/instruction, but it will increase the delay and degrade the performance. Since the objective is to maximize performance for a given power, or to minimize power for a given performance level, using the product
Pareto efficiency is the optimal approach for analyzing trade-offs between different parameters. Pareto efficiency is originally an economic concept. It is defined as: state of economic allocation of resources in which it is impossible to make anyone further better off without making at least one individual worse off. The Pareto frontier is the set of choices that are Pareto-efficient. By focusing on the set of choices that are Pareto-efficient, a designer can make tradeoffs within this set, rather than considering the full range of every parameter. Figure 3.1 demonstrates this concept on results obtained from CACTI simulations done by varying optimization parameter values for a fixed set of main parameter. From the set of values obtained via Pareto frontier, only
the top 15% configurations with close-to-minimal EDP value are selected. Only 15 points from that top 15% were finally selected. Next, regression and optimization is done on those values to get the final equation.

3.1.2 SRAM and Cache Power Model

In the previous section, we gave an overview of the approach used for developing the EDP or delay equation. In this section, we describe the dynamic energy equation used in LibPeq power model. The main parameters for SRAM are: technology generation ($n$), ports ($p$), size ($s$), and bus width ($w$). The main parameters for cache are: technology generation ($n$), size ($s$), associativity ($w$), and line size ($l$). Similar to EDP/delay equation, we have four groups based on SRAM main parameter value range and two groups based on cache main parameter value range. Each group is associated with a different set of coefficient values.

<table>
<thead>
<tr>
<th>Parameter Partitioning</th>
<th>Partition Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 port or 2 ports &amp; (port*size) &lt; 4 KB</td>
<td>Small1</td>
</tr>
<tr>
<td>more than 2 ports &amp; (port*size) &lt; 10 KB</td>
<td>Small2</td>
</tr>
<tr>
<td>1 port or 2 ports &amp; (port*size) &gt; 4 KB</td>
<td>Large1</td>
</tr>
<tr>
<td>more than 2 ports &amp; (port*size) &gt; 10 KB</td>
<td>Large2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Partitioning</th>
<th>Partition Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ways &lt;=4 &amp; size &lt; 64 KB &amp; cacheline &lt; 64 B</td>
<td>Small</td>
</tr>
<tr>
<td>ways &gt; 4 or size &gt;= 64 KB or cacheline &gt;= 64 B</td>
<td>Large</td>
</tr>
</tbody>
</table>

Table 3.1: Partitioning criteria for the power model equation

The equation for computing power is the same as the EDP or delay equation
but with different coefficient values. To derive the energy equation we have divided the 
\( EDP \) equation by the delay equation. The result obtained using this energy equation 
is multiplied with the activity rate to compute power for SRAM and cache-based structures. Equation 3.3 is used for computing power for SRAM and equation 3.4 is used for computing power for cache. Table 3.2 shows the coefficient values used for the two equations. The log in each equation represents natural logarithm, \( E \) stands for dynamic energy and \( a \) to \( g \) are coefficients.

\[
\log(E) = a + b \log(n) + c \log(p) + d \sqrt{s} + \\
e \log(s) + f \sqrt{w} + g \sqrt{w}
\]

\( (3.3) \)

\[
\log(E) = a + b \log(n) + c \sqrt{n} + d \log(w) + \\
e \frac{w}{l} + f \log(s) + g \sqrt{\frac{s}{w \cdot l}}
\]

\( (3.4) \)

<table>
<thead>
<tr>
<th></th>
<th>SRAM</th>
<th>CACHE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Small1</td>
<td>Small2</td>
</tr>
<tr>
<td>( a )</td>
<td>-4.98200</td>
<td>-5.00300</td>
</tr>
<tr>
<td>( b )</td>
<td>2.19600</td>
<td>1.93400</td>
</tr>
<tr>
<td>( c )</td>
<td>0.49610</td>
<td>0.51900</td>
</tr>
<tr>
<td>( d )</td>
<td>-0.00986</td>
<td>0.00934</td>
</tr>
<tr>
<td>( e )</td>
<td>0.54640</td>
<td>0.43500</td>
</tr>
<tr>
<td>( f )</td>
<td>-0.01696</td>
<td>-0.00203</td>
</tr>
<tr>
<td>( g )</td>
<td>0.40270</td>
<td>0.17867</td>
</tr>
</tbody>
</table>

Table 3.2: Coefficients for SRAM and cache energy equations
3.2 Power Model Integration

The main aim of our work is to create a framework in ESESC to incorporate equation-based power model. To explain how the new model was integrated in ESESC, we first give an overview of the existing modeling approach in ESESC. Next, we give details of the LibPeq integration in ESESC.

3.2.1 Existing ESESC Power Model

This section provides an overview of ESESC and the existing approach for modeling processor to compute power. Currently, ESESC relies on CACTI to compute power for SRAM and cache-based structures. As discussed in previous sections, CACTI simulation for full design space sweep is extremely slow and resource intensive.

ESESC calls modified McPAT for power computation, and McPAT makes calls to CACTI to compute power for array-based structures. To understand how the new equation has been integrated, we first need to get an overview of how the processor core is modeled in ESESC. Using McPAT, ESESC is able to model any multicore or manycore processor. The model has basically a three-layer hierarchy. The first level is the architecture level and it represents major processor components such as caches, cores, and memory controller (MC). The second level is the circuit level, in which the building blocks from the architecture level are mapped into major circuit structures such as wires, array-based memory structures, and clock networks (n/w). The third level is the technology level, in which we compute the values of physical parameters (e.g.
 capacittance and resistance) of the wires and of other devices. It uses ITRS parameters for this computation. Figure 3.3 summarizes the hierarchy.

The out-of-order (OoO) processor is modeled with ten pipeline stages, while the in-order processor has only five stages, namely: instruction fetch, instruction decode, execution, memory access, and write back. Figure 3.4 gives a general overview of the ten-stage pipeline used to model OoO processor.

The main units comprising a core are: execution unit (EXU), instruction fetch unit (IFU), load/store unit (LSU), memory management unit (MMU), renaming unit (RU), and an issue/dispatch unit for OoO processors. These units are mapped to corresponding hardware structure. The execution unit contains floating point unit (FPU), arithmetic logic unit (ALU), bypass logic, and register files (RFU).
Figure 3.4: Ten-stage pipeline model for OoO processor

The main components of instruction fetch unit and their respective hardware mappings are:

1. Instruction cache: mapped to array-based structures.

2. Cache controller: comprises of miss buffer, fill buffer, and prefetch buffer. These structures are further mapped to CAM-based structures.


4. Branch Predictor: consists of the branch predictor, the return address stack (RAS), and the branch target buffer (BTB).

The main components of the renaming unit are:

1. Register Alias Table (RAT): comprises of front-end register alias table (FRAT) and rear-end register alias table (RRAT). It is mapped to an SRAM array.

2. Dependency check Logic (DCL): modeled analytically as random logic.

3. Free list buffer (FreeL): mapped to an SRAM array.
The scheduling unit, for OoO processors we have both reservation-station-based and physical-register-file-based architectures. The main components of the Scheduling unit are:

1. Scheduling window: mapped to fully-associative structure with a CAM array and an SRAM array.

2. Reorder buffer (ROB): mapped to a RAM array.


The execution unit shares some structures with the scheduling unit. The main components are:

1. Arithmetic logic unit (ALU): mapped to complex logic model.

2. Floating point unit (FPU): mapped to complex logic model.

3. Register files: mapped to array-based structure SRAM. There are two categories of register files, namely: architectural register file and physical register file. The number of physical registers \( N_{\text{PhysicalRegisters}} \) is a function of number of threads \( N_{\text{threads}} \) and number of architectural register files \( N_{\text{ArchiRegs}} \). Equation 3.5 gives the formula for computing default number of physical register files [13]:

\[
N_{\text{PhysicalRegisters}} = N_{\text{threads}} \times N_{\text{ArchiRegs}} + 100 \quad (3.5)
\]

Number of extra registers is set to 100 as explained by Tullsen et al. [31].
The main components of the memory management unit are:

1. Instruction TLB: mapped to CAM array structure.

2. Data TLB: mapped to CAM array structure.

The main components of the load/store unit are:

1. Data cache: modeled in the same way as the instruction cache of the instruction fetch unit except that it has a write back buffer (WBB). The WBB is divided into a RAM portion and a CAM portion.

2. Load queue and store queue: mapped to CAM array structure.

3.2.2 LibPeq: New EESCC Power Model

In this section, we present how we created the framework to incorporate equation-based power model in EESCC.

3.2.2.1 Configuration File

In EESCC, a configuration file is used to specify various parameters. Here we show a section of configuration file for its format and syntax:

#This is an example configuration file for EESCC

#Specify the type of processor core

coreType = 'tradCORE'

SMcoreType = 'gpuCORE'
# Theses variables are defined here for the GPU-based simulations.

SP_PER_SM = 0

MAXTHREADS = 0

infofile = "foo.info"

# Specify Benchmark for testing processor performance

benchName = "launcher -- stdin crafty.in -- crafty"

# Specify Emulator

cpuemul[0] = 'QEMUSectionCPU'

In the above example, keyword coreType is used for specifying the type of core used in processor, benchName specifies the benchmark that is used for testing the performance and cpuemul[0] is used for specifying the emulator. EESOC reads the entire configuration files and stores in a parser object, called SescConf. It is written in C++ and its member functions allow access to each of these fields within the program [4]. As an example, coreType can be read as SescConf->getCharPtr(coreType). Here getCharPtr(coreType) is a member function of SescConf and coreType is the input argument.

We have used SescConf to read equations for our power model. We have four equations for dynamic power consumption of an SRAM and two equations for a cache, and they are specified in the configuration file as shown here:

[SRAM_Small1]
dynamic = "exp(-4.982+2.196* ln(tech)+0.4961* ln(ports)-0.00986* sqrt(size)"
leakage = "4+C0*V0/V1" #Dummy Leakage Equation

[SRAM_Large1]
dynamic = "exp(-5.446+2.094* ln(tech)+0.886* ln(ports)+0.000458* sqrt(size)
+0.5296* ln(size)-0.011965* width+0.31001* sqrt(width))* (10^(-9))"

leakage = "4+C0*V0/V1" #Dummy Leakage Equation

[SRAM_Small2]
dynamic = "exp(-5.003+1.934* ln(tech)+0.519* ln(ports)+0.009336* sqrt(size)
+0.435* ln(size)-0.002029* width+0.17867* sqrt(width))* (10^(-9))"

leakage = "4+C0*V0/V1" #Dummy Leakage Equation

[SRAM_Large2]
dynamic = "exp(-6.267+1.776* ln(tech)+0.537* ln(ports)+0.00119* sqrt(size)
+0.6389* ln(size)+0.00503* width+0.04* sqrt(width))* (10^(-9))"

leakage = "4+C0*V0/V1" #Dummy Leakage Equation

[CacheEq_Small]
dynamic = "exp(25.7681+7.805* ln(tech)-51.032* sqrt(tech)+0.198*ln(assoc)
-0.533* (assoc/line_size)+0.4692* ln(size)
+0.00395* sqrt(size/(assoc * line_size)))* (10^(-9))"

leakage = "4+C0*V0/V1" #/Dummy Leakage Equation

[CacheEq_Large]
dynamic = "exp( 32.95+8.997* ln(tech)-63.01* sqrt(tech)+0.456 *ln(assoc)
-0.8081* (assoc/line_size)+0.338* ln(size)

37
leakage = "4+C0*V0/V1" #Dummy Leakage Equation

To evaluate the equations used in our power model we have used *muParser*, because of its excellent parsing capability and easy portability [6]. We used *SescConf* to parse the equations as a string and then pass it to *muParser* for evaluation. *muParser* is an extensible high-performance math expression library, which is portable and can be compiled on any standard C++ compiler. It has 25 built-in functions, and 14 predefined operators for computing general math terms such as logarithmic and trigonometric functions. To add more flexibility, it also supports user-defined functions and operators for handling more sophisticated equations. It also allows user-defined numeric and string constants as well as unlimited number of user-defined variables. *muParser* can also be used to implement database queries. Therefore, even if a more complicated equation is developed in future, it can be easily updated in ESESC and parsed using *muParser*.

The C++ code used for parsing is shown below.

```cpp
void SRAM::SetEquation(char* eq_type)
{
  const char *EqExpression;
  if(ports < 3) {
    if(ports*size < 4097) // Size smaller than 4KB
      EqExpression = SescConf->getCharPtr("SRAM_Small1", eq_type);
  else
    // More complicated equations
  }
```

\[ +0.0062 \times \sqrt{\frac{\text{size}}{\text{assoc} \times \text{line_size}}} \times 10^{-9} \]
EqExpression = SescConf->getCharPtr("SRAM_Large1",eq_type);
}

else {
    if(ports*size < 10241) // Size smaller than 10KB
        EqExpression = SescConf->getCharPtr("SRAM_Small2",eq_type);
    else
        EqExpression = SescConf->getCharPtr("SRAM_Large2",eq_type);
}

if(strcasecmp(eq_type, "dynamic") == 0 ) {
    // Declare Variables of dynamic equation and link to object variables
    // Since muParser only takes variables as double. All variables should be
    // declared as double
    dynamic_eq.DefineVar("tech",&tech);
    dynamic_eq.DefineVar("ports",&ports);
    dynamic_eq.DefineVar("width",&width);
    dynamic_eq.DefineVar("size",&size);
    dynamic_eq.SetExpr((string_type)EqExpression);
}

else
    leakage_eq.SetExpr((string_type)EqExpression);
In the above code, first an equation for SRAM is obtained based on its size and number of ports using the SescConf object. Next, equation variables (technology, ports, width, and size) are defined and linked using the DefineVar function of the muParser object. After that we set the expression using the SetExpr function. We use a similar approach for a cache, and the same approach can be extended for other structures such as CAM.
Chapter 4

Performance Evaluation

In this chapter, we evaluate our new power model in terms of speed. For speed gains, we focus on time-saved during power model initialization and on time taken to run a simulation for more than a billion instructions. We also show a comparison of LibPeq and orginal ESESC results for dynamic power of SRAM and cache-based structures. All simulations are performed on a 64-bit AMD Opteron Processor 6172. It has 48 cores (6 processors), 128 GB memory, 512 KB cache and 2.1 GHz speed. We took the average of three simulation runs for generating any plot. The main focus of our work is to provide a framework to incorporate equation-based power model to ESESC. Since, the accuracy of the model is a function of the accuracy of the equation used, therefore it has not been discussed in our thesis. Discussion on correctness of the EDP and delay equation is present in Ebrahimi MS thesis [10]. Original ESESC uses McPAT version 0.6 and CACTI version 6.5.
4.1 Speed Gain

For any good microprocessor simulator, speed is of great concern as most research groups work with tight deadlines and limited computing budget. Simulators' capability to run a benchmark is thousands of times slower than the actual processor. Moreover, power modeling tools like CACTI further slow down the simulation since, even default settings need to perform multiple iterations through structural parameters to get an optimal configuration. Also, for complex software such as CACTI and McPAT, the initialization time is one of the sources of high latency.

In order to evaluate speed gain, we observe the difference in time to initialize the original ESESC power model with the time taken to initialize LibPeq. Figure 4.1
shows the comparison between initialization times for two power models. Initialization
time does not depend on number of instructions for which the model was run. LibPeq
took only 0.04 seconds to initialize while the original ESESC power model using CACTI
took 160.43 seconds. All the configurations for both simulations were kept identical.
By using LibPeq we reduce the initialization time by 99.98%. This can be primarily
attributed to the fact that LibPeq is a simple statistical power model while CACTI is
a very sophisticated power model with numerous sub-components and threads. Initial-
ization time for CACTI simulations in debug mode can take hours and using LibPeq we
can achieve significantly better simulation time. Major part of time saved using LibPeq
can be attributed to the speed gained during initialization phase. However, we have
also generated plots to see how LibPeq simulations fair with respect to original ESESC
for large instruction size.

Real time or wall clock time refers to total elapsed time, including time used by
other processes/threads and time the process remains blocked. To evaluate with respect
to real time, data was collected by running simulation for two billion and 15 billion
instructions. In these simulations, we skipped first one billion instructions and simulated
one billion and 14 billion instructions. When we run LibPeq and original CACTI-based
ESESC for two billion instructions, we find that LibPeq is 16.81% (LibPeq = 193 s and
CACTI = 232 s) faster than the original ESESC. CACTI’s real time is much higher
because it has numerous modules and threads, and to get the optimal configuration it
has to search a huge design space. Threads usually correspond to parallelism and speed
but, as Amdahl’s law states, “The speedup of a program using multiple processors
Figure 4.2: Real time for two billion instructions

Figure 4.3: Real time for 15 billion instructions
in parallel computing is limited by the time needed for the sequential fraction of the program”. The key here is that even though CACTI has multiple threads, the sequential parts result in situations where threads for one step in the algorithm have to wait for the threads of the previous step to signal that they are ready. The more sequential parts there are in a program, the less benefit it will have from multiple cores. So, having multithreaded complex software does not always give desired level of speedup. Figure 4.2 and Fig. 4.3 show the real time plot for simulation with two billion and 15 billion instructions, respectively.

Also, it is important to note that the LibPeq analytical model for SRAM and cache was developed using thousands of CACTI simulations that covered full design space. While developing the model, some of the simulations for a large SRAM size (1 MB) took 3-10 days to do a full sweep across all optimization parameters. By default, when McPAT calls CACTI in ESESC, CACTI does not perform a full sweep of design search space. Therefore, a typical processor configuration requires 5-6 minutes to estimate energy using ESESC with McPAT and CACTI. A few minutes seem a small delay for a simulation, but it is noticeable amount of time while debugging or developing the design. If the CACTI simulations were to be extended to consider exhaustive exploration of design search space, it would require several hours of initialization for each run. Therefore, with this new analytical model, the power computation becomes very straightforward and quick. Overall, using LibPeq in ESESC gives us significant gains in design turnaround time.
4.2 Comparison With Original ESESC

This section provides comparison between LibPeq results and original ESESC results. Since our equations are only valid for computing dynamic power for SRAM and caches, we have not compared the values for CAM-based structures and total power.

![Figure 4.4: Renaming unit](image)

The plots in Fig. 4.4 to 4.8 show the dynamic power comparison for SRAM and cache-based structures. Using LibPeq the minimum difference in percentage with respect to CACTI is 13.81% for Icache and the maximum difference is 17.16% for ROB. This difference in value does not reflect the accuracy of LibPeq because, the EDP and delay equations were derived after doing a full sweep of CACTI search space and is expected to be different from the default simulation results of CACTI.
Figure 4.5: Instruction Fetch Unit

Figure 4.6: Register Files
Figure 4.7: Reorder Buffer

Figure 4.8: Icache
Chapter 5

Conclusion

In order to speed up the power computation in the ESESC simulator, we have replaced CACTI with a new equation-based power model: LibPeq. Processor design trend is towards higher frequency and more complex architectures, resulting in high power and heat dissipation issues. Most modern architectural simulator support energy model extension, but simulation speed has always been an issue. ESESC with LibPeq enables microprocessor-researchers to compute power consumption estimates in a shorter time.

LibPeq uses a set of regression-based equations for computing power. The energy equations were derived from Pareto-optimal $EDP$ and delay equation for SRAM and cache-based structures. To model $EDP$ or delay equation, first the CACTI default search space was expanded by running CACTI for all possible optimization values for a given set of main parameters. Then, Pareto-optimal energy and delay values were selected from the Pareto frontier. After that, regression-based models were derived to
select best configuration points with optimal $EDP$. The optimized regression equation simplify the modeling since it captures the trends based on data obtained from thousands of CACTI simulations for every set of main parameters. The equation generated can be used across different technologies, making it simpler and faster to do the evaluation of SRAM and cache-based components.

We have shown that by integrating equation-based power model to ESESC, we get an average speed gain of 16.8% for large simulation runs and 99.98% improvement in power model initialization time. Using equation-based power computation covers more search space than the default CACTI simulation used in ESESC. Although possible, in reality, CACTI instantiations do not cover the whole search space due to resource and time constraints. ESESC simulations with default CACTI settings take 4-6 minutes but, running CACTI simulations to consider the entire search space takes hours.

5.1 Future Work

Future work can involve developing analytical models for CAM/fully associative structures. The same approach used for SRAM and cache structures can be applied to develop CAM equation and to integrate it with ESESC. Another aspect that needs to be addressed in the near term is to add an equation for computing leakage power. Once we have the equation for computing leakage power, we can easily update it in our LibPeq framework. Currently, we are using a dummy equation to compute leakage power for SRAM and cache-based structures. Also, another improvement is to update
the McPAT version (0.6) used in ESESC. McPAT 0.8 has many new features that can further improve the performance of ESESC. In the future we can implement LibPeq framework for other architectural simulators that use energy models such as McPAT and CACTI.
Bibliography


[27] Subbarao Palacharla, Norman P. Jouppi, and James E. Smith. Quantifying the


[33] Thomas F. Wenisch, Roland E. Wunderlich, Babak Falsafi, and James C. Hoe.
