AN EFFICIENT DESIGN SPACE EXPLORATION FRAMEWORK TO OPTIMIZE POWER-EFFICIENT HETEROGENEOUS MANY-CORE MULTI-THREADING EMBEDDED PROCESSOR ARCHITECTURES

by

Kushal Datta

A dissertation submitted to the faculty of
the University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree Of Doctor of Philosophy In
Electrical and Computer Engineering

Charlotte

2011

Approved by:

Dr. Arindam Mukherjee

Dr. Arun Ravindran

Dr. Bharat S. Joshi

Dr. Kazemi Mohammed

ABSTRACT

KUSHAL DATTA. An efficient design space exploration framework to optimize power-efficient heterogeneous many-core multi-threading embedded processor architectures. (Under the direction of DR. ARINDAM MUKHERJEE)

By the middle of this decade, uniprocessor architecture performance had hit a roadblock due to a combination of factors, such as excessive power dissipation due to high operating frequencies, growing memory access latencies, diminishing returns on deeper instruction pipelines, and a saturation of available instruction level parallelism in applications. An attractive and viable alternative embraced by all the processor vendors was multi-core architectures where throughput is improved by using micro-architectural features such as multiple processor cores, interconnects and low latency shared caches integrated on a single chip. The individual cores are often simpler than uniprocessor counterparts, use hardware multi-threading to exploit *thread-level parallelism* and *latency hiding* and typically achieve better performance-power figures. The overwhelming success of the multi-core microprocessors in both high performance and embedded computing platforms motivated chip architects to dramatically scale the multi-core processors to many-cores which will include hundreds of cores on-chip to further improve throughput. With such complex large scale architectures however, several key design issues need to be addressed. First, a wide range of micro-architectural parameters such as L1 caches, load/store queues, shared cache structures and interconnection topologies and non-linear interactions between them define a vast non-linear multi-variate micro-architectural design space of many-core processors; the traditional method of using extensive in-loop

simulation to explore the design space is simply not practical. Second, to accurately evaluate the performance (measured in terms of cycles per instruction (CPI)) of a candidate design, the contention at the shared cache must be accounted in addition to cycle-by-cycle behavior of the large number of cores which superlinearly increases the number of simulation cycles per iteration of the design exploration. Third, single thread performance does not scale linearly with number of hardware threads per core and number of cores due to memory wall effect. This means that at every step of the design process designers must ensure that single thread performance is not unacceptably slowed down while increasing overall throughput. While all these factors affect design decisions in both high performance and embedded many-core processors, the design of embedded processors required for complex embedded applications such as networking, smart power grids, battlefield decision-making, consumer electronics and biomedical devices to name a few, is fundamentally different from its high performance counterpart because of the need to consider *(i)* low power and *(ii)* real-time operations. This implies the design objective for embedded many-core processors cannot be to simply maximize performance, but improve it in such a way that overall power dissipation is minimized and all real-time constraints are met. This necessitates additional power estimation models right at the design stage to accurately measure the cost and reliability of all the candidate designs during the exploration phase.

In this dissertation, a statistical machine learning (SML) based design exploration framework is presented which employs an execution-driven cycle-

accurate simulator to accurately measure power and performance of embedded many-core processors. The embedded many-core processor domain is *Network Processors* (NePs) used to processed network IP packets. Future generation NePs required to operate at terabits per second network speeds captures all the aspects of a complex embedded application consisting of shared data structures, large volume of compute-intensive and data-intensive real-time bound tasks and a high level of task (packet) level parallelism. Statistical machine learning (SML) is used to efficiently model performance and power of candidate designs in terms of wide ranges of micro-architectural parameters. The method inherently minimizes number of in-loop simulations in the exploration framework and also efficiently captures the non-linear interactions between the micro-architectural design parameters. To ensure scalability, the design space is partitioned into *(i) core-level micro-architectural parameters* to optimize single core architectures subject to the real-time constraints and (ii) *shared memory level micro-architectural parameters* to explore the shared interconnection network and shared cache memory architectures and achieves overall optimality. The cost function of our exploration algorithm is the total power dissipation which is minimized, subject to the constraints of real-time throughput (as determined from the terabit optical network router line-speed) required in IP packet processing embedded application.

ACKNOWEDGMENTS

First of all, I thank my parents Mr. Kalyan Kumar Datta and Mrs. Tamali Datta for providing me love, strength, support, inspiration, knowledge, wisdom and independence. I feel an endless sense of pride and emotion as I acknowledge and appreciate their contribution in my life. I feel equally blessed to have my elder sister Mrs. Kabita (Datta) Hazra and her beloved husband Mr. Susovan Hazra and thank them from the bottom of my heart for providing me support and friendship all throughout my life no matter how dire the situation. I also would like to thank my late grandmother Mrs. Kamala Datta and her sister-in-law Mrs. Bimala Bala Datta for being such a big part of my life during my adolescent years – loving me endlessly and teaching me to confront every adversity with pride, faith and love. I also thank my late uncle Mr. Kuber Datta for teaching me to be enthusiastic and full of life and freedom.

I would like to thank the rest of my family, all my cousin brothers and sisters, especially Kaushik Dutta and my dear friends Jong-ho Byun, Aby Kuruvilla and Rewa S. Tikekar, who have been there through my toughest times and knocked sense into me every now and then.

I would especially like to thank my advisor Dr. Mukherjee for being my friend, philosopher and guide. He will always be a source of inspiration to me. I also thank my committee members, Dr. Arun Ravindran and Dr. Bharat S. Joshi for their continuous support, for reviewing my work and providing me with useful feedback.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Embedded Many-core Processors

Recent years have witnessed a dramatic transition in the complexities and capabilities of embedded processors. Examples include Cisco 40 core Quantum Flow network processor [1], Freescale QorIQ series network processors with upto to 8 cores [2], Netlogic XLP316L 16 core quad-issue processor with 4 hardware threads per core [3], Netronome Network Flow Processor with 40 IXP cores with 4-threads per core [4], NVIDIA Tesla 10 core GGPGPU with 24 scalar stream processors per core [5], and PicoChip 250-300 core picoArray digital signal processor [6]. Similar to high performance processor vendors, those in the embedded domain are permanently altering their existing roadmaps to incorporate hundreds of cores on the same chip in the coming decade – the embedded many-core processor. However, embedded computing is fundamentally different from its high performance counterpart because of the need for low energy and real-time operation required in complex embedded applications such as networking, smart power grids, battlefield decision-making, consumer electronics and biomedical devices, to name a few. To satisfy these performance requirements, conceivably the future embedded many-core processor will have hundreds of heterogeneous cores on chip, some of which will be fine grained multi-threaded RISC cores to exploit embedded task level

parallelism, and some highly application-specific cores – all connected to hierarchies of distributed on-chip memories by high speed networks-on-chip (NoCs). While the industry focus is on putting higher number of cores on a single chip, the key challenge is to optimally architect these embedded many-core processors for low energy operations while satisfying area and often stringent real-time constraints. With such complex many-core architectures, the traditional approach to processor design through extensive simulations is no longer viable due to the large design space that must be explored in-order to optimize power-performance.

Future generation embedded applications are expected to grow even more complex consisting of a large volume of computational and data intensive real time bound tasks sharing large data structures. To methodically study the power-performance trade-offs of embedded many-core processors to be designed to satisfy the requirements of such complex embedded applications, we focus on Network Processors (NePs) executing the functions of *IP packet processing* as the representative processor domain. Our idea is to thoroughly investigate the high degree of task level parallelism, shared data structures and real-time operations present in packet processing application and establish a modeling and design exploration framework for NePs in this dissertation. The methodology can be easily extended to design complex embedded and high performance many-core platforms.

1.2 Demands of High Performance Packet Processing Routers

Internet demand is growing at an explosive rate. A large volume of

technology consumer products such as personal computers, workstations, web-enabled mobile devices and multimedia-enabled smart-phones are used to connect to various websites on a regular basis. Also, an increased use of websites offering online voice and video services such as Hulu, Youtube and Facebook to name a few, has resulted in a surge in overall network traffic. The total network traffic in North America (the highest IP-traffic generating region) is predicted to be approximately 19.0 exabytes per month by 2014 [7]. With such an explosive increase in data demand, existing *edge routers* used to interface between different communication networks and *core routers* which constitute the backbone of the internet, are identified to be the bottlenecks in the next generation ultra high speed networks [8]. The Network Processors (NePs) powering these routers can support maximum line speeds of 10 to 100 gigabits per second [9-14], which is insufficient for handling the predicted volume of data in the future. Power is also a critical concern in the design of high performance NePs. Cost is increased by the requirements of larger power supplies and cooling systems. Reliability is compromised by thermal hot-spots on chip. Power increase also adversely affects operating environment features by driving higher utility costs and higher installation and maintenance costs. Cool running NePs pack more ports into a smaller space within thermal operating limits, and have the capability of staying online longer in a battery back-up mode when main power fails. As a result, next generation NePs must be architected to achieve throughput that can support terabit per second (TBPS) line speeds, and yet operate under low power budgets so that the overall operating cost can be

minimized and reliability can be improved.

NePs execute real-time Internet Protocol (IP) packet processing applications, which consist of compute-bound and data-bound tasks [15-17]. Compute-bound tasks include cyclic redundancy error checking codes, block-ciphering and likewise. Data-bound tasks include traffic monitoring, IP table lookups, packet fragmentation, Reed Solomon's error checking codes, deep packet inspection and others. Incoming packets in a router are classified as either high priority hard real-time constrained conversational voice packets for example, or lower priority soft real-time constrained non-critical video and other content-delivery packets [18]; the incoming packets are scheduled on the NePs according to their priorities. Once error-checking and route calculations are completed, the packets are sent to the outward queues. Two critical shared data structures in this system are the routing table and the traffic monitoring table. The routing table contains millions of forward route entries which are read by incoming packets to look-up the next destinations. It is rarely updated. On the other hand, the traffic monitoring database is updated with the details of every incoming packet.

1.3 Micro-architectural Domain of Network Processors

Existing high-performance network processors are based on the following micro-architectures: *superscalar* (SS), *streaming single instruction multiple data* (S-SIMD), *chip multi-processor* (CMP), and *simultaneous multi-threading* (SMT) [10, 11, 14, 19]. While SS exploits instruction level parallelism (ILP), it does not take advantage of the high degree of task (packet) level parallelism (TLP)

inherent in IP packet processing. S-SIMD implements a systolic array of packet processing kernels and the packet data is streamed from one stage to another. However the benefit of pipelining of the packet operations is mitigated by stalls encountered at the shared data structure read/write stage for every incoming packet. Although network processors designed with SMT are able to process packets with high throughput and meet real time constraints, they have high power dissipation and hence are not always cost-effective. Commercial network processor architectures combine these paradigms along with ASIC acceleration engines. For example, EzChip's TopCore technology uses an array of superscalar processors with customized instruction sets [20]; Intel's Next Generation Microengine Architecture combines CMP and multithreading along with inter-processor pipelined operation using next neighbor registers [21]; Netronome's NFP-3240 network flow processor is an array of 40 1.4GHz micro-engine RISC processor [4].

1.4 Dissertation Contribution

Our design philosophy to achieve a low power TBPS network processor is to use shared memory many-core architecture. Low latency on-chip shared cache memories helps us to minimize off-chip accesses as the large shared data structures (IP lookup table and Traffic monitoring table) are read or updated for all packets. All the processor cores are in-order and use hardware multi-threading; the thread selection policy is fine-grained multi-threading (FGMT). In-order FGMT [22] utilizes simple six stage pipeline shared between the hardware threads, enabling us to achieve *(i)* high throughput per-core by *latency hiding* and

(ii) minimize the power dissipation of a core by avoiding complex micro-architectural structures such as instruction issue queues, re-order buffers and history-based branch predictors typically used in superscalar or other types of hardware multi-threading techniques. Also, to achieve better power-performance points we make the processor cores structurally heterogeneous. This way more hardware resources are invested into processor cores designed to compute more resource-hungry tasks and overall on-chip hardware resources are optimally utilized. Dynamic power-saving mechanisms such as power-gating and dynamic voltage and frequency scaling (DVFS) are used at the core level to minimize power dissipation in case of idle cores. Inside the cores, clock-gating is enabled at all pipeline stages to minimize dynamic power dissipation. A high level of packet-level parallelism is achieved due to the large number of cores, which also overcomes the well-known power wall problem.

In this dissertation we present an efficient and scalable statistical machine learning based design space exploration framework. Our first step includes the design and development of an instruction trace-driven cycle-accurate many-core processor simulator used to measure throughput (in terms of cycles per instruction) of candidate many-core designs for different combinations of various micro-architectural parameters belonging to this design space. The simulator called **C**hip Multi-threading **A**rchitecture **S**imulator for **P**erformance **E**nergy and A**R**ea Analysis (CASPER) is a SPARCV9 instruction set based processor simulator. To simultaneously measure power dissipation of candidate designs along with throughput, CASPER is empowered with power estimation models of

each micro-architectural block enabling us to accurately measure power dissipation every cycle. Our literature survey of existing functional and cycle-accurate multi-core simulators and network processor simulators in Chapter 2 show that to the best of our knowledge no such large scale simulation platform exist which can accurately measure power and performance of many-core designs cycle-by-cycle. In addition, a well-established Solaris 5.10 software stack on top of CASPER enables us to execute any embedded or high performance application on this simulation platform.

Once we have a validation platform, our second step is to apply a divide and conquer method to explore the design space in a stepwise fashion. Our many-core micro-architectural design space is defined by the core-level parameters which include level one (L1) instruction and data (I/D) cache sizes and number of hardware threads per core, pipeline depth, I/D miss queues and store buffers. The chip-level parameters include number of cores, interconnection architecture, shared second level memory (L2) queue size, L2 organization and access times. Although all of the above micro-architectural parameters are tunable in CASPER to simulate different configurations, it is not practical to use in-loop simulation while exploring the vast micro-architectural design space. To resolve this issue we first optimize the core architectures. The objective of this step is to design a core in such a way that it processes a packet within the real-time boundary and the power dissipation is minimized. Several packet types exist according to which the sequence of functions used to process a packet varies. Hence the micro-architecture of a core optimized for a particular packet type also

varies from other cores designed for other packet types. Using linear statistical regression, the power and performance regression models of the cores are derived using randomly chosen values of the core-level micro-architectural parameters. Once the models are derived, they are used instead of in-loop simulation in a Genetic Algorithm based heuristic to find optimal core micro-architectures for all packet types.

At this point of our exploration, chip-level parameters still not been used. Our third step involves core interaction modeling and shared cache optimization. We estimate the number of cores required for processing a particular distribution of packet types. For a given choice of the interconnection network (for example, crossbar), we build a predictive model for the contention (and hence the associated L2 cache access time) and power dissipation, and the L2 cache banks. The predictive models are built from training data obtained through the macro-simulator L2MacroSim implemented in CASPER. Only the core to L2 cache and L2 cache to memory reply/acknowledgement packets are simulated. The inputs to the L2 MacroSim are L2 cache input queue size per core, cache bank size, line size, associativity, number of L2 banks, L1 I and D cache sizes, line sizes and associativities and instruction trace files for each thread in each core. The individual core parameters are set to their optimal values from previous step. The L2MacroSim enables significant savings in simulation time while capturing the interaction between the cores. The predictive models for core interaction are used to optimize the power dissipation of the L2 cache banks while satisfying the real-time constraints. If the L2 access time constraints cannot

be satisfied, we choose the next best core for each packet type and repeat steps two and three.

The rest of the dissertation is organized as follows. Chapter 2 describes existing processor simulators and architecture exploration algorithms. Chapter 3 explains the embedded network packet processing benchmark which we use in this research. Chapter 4 and 5 discusses the structural details and organization of a many-core processor simulator CASPER. Our exploration algorithm is elaborated in Chapter 5 and 6. Results of our research are presented and analyzed in Chapter 6, and finally in Chapter 7 we present our conclusions.

CHAPTER 2: BACKGROUND

2.1 Processor Simulators

Virtutech Simics [23] is a full-system scalable functional simulator for embedded systems. The released versions support microprocessors such as PowerPC, x86, ARM and MIPS. Simics is also capable of simulating any digital device and communication bus. The simulator is able to simulate anything from a simple CPU + memory, to a complex SoC, to a custom board, to a rack of multiple boards, or a network of many computer systems. Simics is empowered with a suite of unique debugging toolset including reverse execution, tracing, fault-injection, checkpointing and other development tools. Similarly, Augmint [24] is an execution-driven multiprocessor simulator for Intel x86 architectures developed in University of Illinois, Urbana-Champagne. It can simulate uniprocessors as well as multiprocessors. The inflexibility in Augmint arises from the fact that the user needs to modify the source code to customize the simulator to model multiprocessor system. However both Simics and Augmint are not cycle-accurate and they model processors which do not have open-sourced architectures or instruction sets; this limits the potential for their use by the research community. Another execution-driven simulator is RSIM [25] which models shared-memory multiprocessors that aggressively exploit instruction-level parallelism (ILP). It also models an aggressive coherent memory system and

interconnects, including contention at all resources. However throughput intensive applications which exploit task level parallelism are better implemented by the fine-grained multi-threaded cores that our proposed simulation framework models. Moreover we plan to model simple in-order processor pipelines which enable thread schedulers to use small-latency, something vital for meeting real-time constraints.

General Execution-driven Multiprocessor Simulator (GEMS) [26] is an execution-driven simulator of SPARC-based multiprocessor system. It relies on functional processor simulator Simics and only provides cycle-accurate performance models when potential timing hazards are detected. GEMS Opal provides an out-of-order processor model. GEMS Ruby is a detailed memory system simulator. GEMS Specification Language including Cache Coherence (SLICC) is designed to develop different memory hierarchies and cache coherence models. The advantages of our simulator over the GEMS platform include its ability to *(i)* carry out full-chip cycle-accurate simulation with guaranteed fidelity which results in high confidence during broad micro-architecture explorations, and *(ii)* provide *deep chip vision* to the architect in terms of chip area requirement and run-time switching characteristics, energy consumption, and chip thermal profile.

SimFlex [27] is a simulator framework for large-scale multiprocessor systems. It includes (a) Flexus – a full-system simulation platform and (b) SMARTS – a statistically derived model to reduce simulation time. It employs systematic sampling to measure only a very small portion of the entire application

being simulated. A functional model is invoked between measurement periods, greatly speeding the overall simulation but results in a loss of accuracy and flexibility for making fine micro-architectural changes, because any such change necessitates regeneration of statistical functional models. SimFlex also includes FPGA-based co-simulation platform called the ProtoFlex. Our simulator can also be combined with an FPGA based emulation platform in future, but this is beyond the scope of this work.

MPTLsim [28] is is a uop-accurate, cycle-accurate, full-system simulator for multi-core designs based on the X86-64 ISA. MPTLsim extends PTLsim [29], a publicly available single core simulator, with a host of additional features to support hyperthreading within a core and multiple cores, with detailed models for caches, on-chip interconnections and the memory data flow. MPTLsim incorporates detailed simulation models for cache controllers, interconnections and has built-in implementations of a number of cache coherency protocols.

NePSim2 [30] is an open source framework for analyzing and optimizing NP design and power dissipation at architecture level. It uses a cycle-accurate simulator for Intel's multi-core IXP2xxx NPs, and incorporates an automatic verification framework for testing and validation, and a power estimation model for measuring the power consumption of the simulated NP. To the best of our knowledge, it is the only NP simulator available to the research community. NePSim2 has been evaluated with cryptographic benchmark applications along with a number of basic testcases. However, the simulator is not readily scalable to explore a wide variety of NP architectures.

McPAT [31] is an integrated power, area and timing modeling framework for multi-core and many-core architectures. At the core level it includes models of micro-architectural components such as in-order, out-of-order processor cores while at the chip level it consists of shared caches, multiple clock domains, memory controllers and NoC. The critical path timing models, area models and leakage power model at the circuit level enables McPAT to estimate power dissipation of a simulated design. However, McPAT is a static power dissipation model and does not contain any cycle-accurate behavior.

Although the available processor simulators are effective for exploring different micro-architectural design spaces, CASPER provides us the flexibility to interchangeably tune impactful micro-architectural parameters such as number of threads in a core, pipeline depth, multiple clock domains, number of cores, interconnection network, shared L2 cache size, associativity and line size. Such a wide range of tunable parameters are not found in other simulators. Also, none of the available simulators provide power estimation for simulated designs. The built-in scalable HDL models of all the micro-architectural blocks in our design such as arithmetic unit, queues, caches and arbiters along with technology libraries ranging from 90nm to 22nm are used to accurately model delay, dynamic and leakage power in CASPER. This is an extremely powerful feature enabling us to accurately measure power dissipation of candidate designs right at the design stage. A stripped down version of the Solaris 5.10 OS kernel is ported onto CASPER which enables us to study a wide range of high performance embedded benchmarks. The details of the simulator and micro-

architectural features are described in Chapter 4.

2.2 Multi- and Many-core Design Space Exploration

Exploring the many-core processor design space through exhaustive cycle-accurate simulation is not practical due to the prohibitively long simulation time and its superlinear increase as the numbers of cores are scaled. Several techniques have been proposed that avoids exhaustive simulations in effectively exploring the uniprocessor [32-35] and many-core [36-38] design space. We first review recent research on modeling and exploring multi- and many-core architectures.

Lee et al. [36] minimize many-core simulation times in estimating performance through composable regression models for baseline uniprocessor performance, cache contention, and delay penalty. Their unicore simulation platform is an execution driven, cycle accurate IA-32 simulator modeling a superscalar, out-of-order architecture. Long instruction traces derived from a variety of application areas ranging from digital home to the server are used as benchmarks. The uniprocessor regression model predicts the baseline performance of each core while the contention regression model predicts interfering accesses to shared resources from other cores. Uniprocessor and contention model outputs are composed in a penalty regression model that considers the contention as a secondary penalizing effect. A trace simulation is stated to be sufficient for developing the contention and penalty models, thus greatly reducing the overall simulation time. A median CPI error of 6.6% is reported for quad-core processors. The major advantage of their work is the

scalability of the methodology to hundreds of cores. The authors have only focused on developing regression models for predicting CPI and not for power estimation.

Ipek et. al. [37] use artificial neural networks to predict performance of a multi-core processor using a small sized training set drawn from the processor design space. Partial simulation techniques based on SimPoint where only certain application intervals or simulation points are modeled, are employed to reduce the simulation time. Benchmarking applications are derived from the SPEC OMP and parallel NAS benchmarks. An average predicted IPC error of 4-5% is reported when the neural network is trained using a 1% sample drawn from a multi-core design space of 8 cores with 250K points and up to 55× performance swings among different system configuration. Similar to Lee et. al. the authors do not model processor power dissipation. More importantly, the authors do not consider chip level shared micro-architectural components such as shared L2 cache and interconnect network which may critically affect performance and power due to the contention in the shared resources. Kang and Kumar [39] treat the multi-core processor design space exploration problem as a classic search and optimization problem with a simulation-in-the-loop approach and use of a rule based machine learning algorithm to prune the search space. The optimization algorithms include steepest ascent hill climbing and genetic algorithms. The machine learning algorithms includes 1-tuple tagging based on the complexity of the cores (simple, moderate, and complex), and 5-tuple tagging based on architecture parameters (Simple, D-cache intensive, I-Cache intensive,

Execution units intensive, and Fetch Width intensive). The objective functions for the optimizations are performance, power, and area. Simulations are done using a modified version of SMTSim. Power and area estimates are obtained for different hardware structures from existing literature. The benchmarks are drawn from SPEC2000, IBS, Olden, and Mediabench. The authors report that their search/machine learning approach achieves within 1% of the performance compared to an exhaustive simulation approach for a 4 core system while being 3800 times faster. However, similar to Ipek et. al. the authors do not consider chip level shared micro-architectural components. Also, their power estimation approach does not allow the study of the dependence of power dissipation on architectural parameters. Regarding exploration of network processor architectures, Wolf and Tillman [40] present an analytical model performance model for predicting the performance, chip area, and power consumption for a prototype network processor parameterized using the Commbench network processing benchmark; Mysore et. al, [41] propose a sensor network benchmark, WiSeNBench,and use an ARM simulator to identify some of the key characteristic behaviors; Lin et. al, [42] use a combination of analytical models and simulations to explore core-centric network processor architectures; Salehi et. al, [43] optimize of a superscalar MIPS network processor through exhaustive simulation. Modeling many-core architecture with an analytical approach requires many simplifying assumptions about the architecture while simulations-only approach suffers from the drawbacks mentioned earlier. Dubach et. al. [33] presents an approach that co-designs an optimizing compiler and architecture

using a machine learning approach. Their framework consists of the Xtrem simulator for the Intel XScale architecture, gcc for the compilier, MiBench for the benchmark, and Support Vector Machines (SVM) for modeling the design space. The best design achieves significant performance increases resulting in a 13% improvement in execution time, 23% savings in energy and an energy-delay product (ED) of 0.67. However, their work is limited to unicore processor architectures. Although, our methodology can incorporate compiler optimizations, these optimizations alone may not achieve sufficient performance on many-core processors.

2.3 Statistical Machine Learning for System Optimization

Statistical machine learning (SML) algorithms can be used to model multivariate data sets. The basic framework in machine learning based optimization includes tunable specification, observables identification, training data collection and data analysis. Brewer [44] uses a linear regression to select the best data partitioning scheme for a given problem size; Vuduc [45] employs support vector machines to construct a non-parametric model of the shape of the partitions of the input space of sparse matrix kernels; Cavazos et. al. [46] use a logistic regression model to predict the optimal set of compiler flags for the SPEC benchmark suite; Ganapathi et. al. [47] use Kernel Canonical Correlation Analysis to effectively identify the relationship between a set of optimization parameters and a set of resultant performance metrics to explore the search space for stencil algorithms; Liao et. al. [48] evaluate several classical machine learning algorithms such as Nearest Neighbor, Naive Bayes, Decision Tree,

Support Vector Machines, Multi-layer perception and Radial Basis Function to optimize pre-fetch configurations for data center applications; Li et. a. [49] use machine learning based online performance prediction for runtime parallelization and task scheduling; Leather et. al. [50] develop a new technique to automatically generate good features for machine learning based optimizing compilation by improving the quality of a machine learning heuristic through genetic programming and predictive modeling. The successes of the above listed research efforts indicate the power of machine learning in directing program and system optimization.

CHAPTER 3: EMBEDDED NETWORK PROCESSING BENCHMARK (ENEPBENCH)

To evaluate the performance and power dissipation of candidate designs we have developed a benchmark suite called *Embedded Network Packet Processing Benchmark* (ENePBench) which emulates the IP packet processing tasks executed in a network router. The router workload varies according to internet usage where random number of IP packets arrive at random intervals. To meet a target bandwidth, the router has to *(i)* process a required number of packets per second and *(ii)* process individual packets within their latency constraints. The task flow is described in Figure 3-1. Incoming IPv6 packets are scheduled on the processing cores of the NeP based on respective packet types and priorities. Depending on the type of a packet different header and payload processing functions process the header and payload of the packet respectively. Processed packets are either routed towards the outward queues (in case of pass-through packets) or else terminated.

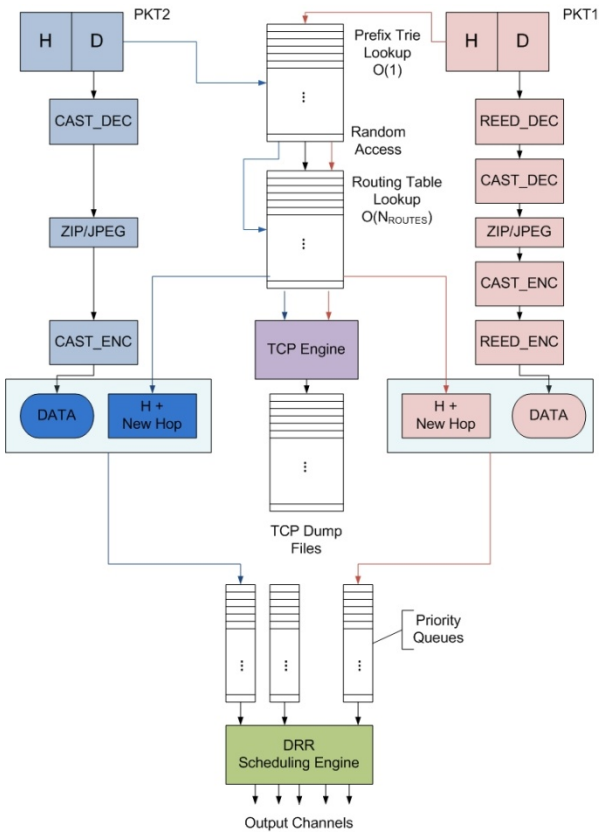

Figure 3-1: Pictorial representation of IP packet header and payload processing in two packet instances of different types

The packet processing functions of ENePBench are adapted from CommBench 0.5 [51]. Routing table lookup function RTR, packet fragmentation function FRAG and traffic monitoring function TCP constitute the packet header functions. Packet payload processing functions include encryption (CAST), error detection (REED) and JPEG encoding and decoding as shown in Table 3-1.

Table 3-1: ENePBench: Packet processing functions

Function Type	Function Name	Description
Header Processing Functions	**RTR**	A Radix-Tree routing table lookup program

	FRAG	An IP packet fragmentation code
	TCP	A traffic monitoring application
Payload Processing Functions	**CAST**	A 128 bit block cipher algorithm
	REED	An implementation of Reed-Solomon Forward Error Correction scheme.
	JPEG	A lossy image data compression algorithm.
Packet Scheduler	**DRR**	Deficit Round Robin fair scheduling algorithm

Functionally, IP packets are further classified into types TYPE0 to TYPE4 as shown in Table 3-2. The headers of all packets belonging to packet types TYPE0 to TYPE4 are used to lookup the IP routing table (RTR), managing packet fragmentation (FRAG) and traffic monitoring (TCP). The payload processing of the packet types, however, is different from each other. Packet types TYPE0, TYPE1 and TYPE2 are compute bound packets and are processed with encryption and error detection functions. In case of packet type TYPE3 and TYPE4, the packet payloads are processed with both compute bound encryption and error detection functions as well as data bound JPEG

encoding/decoding functions.

Table 3-2: Packet Types used in ENePBench

Packet Type	**Header Functions**	**Data Functions**	**Characteristic**	**Type of Service**
TYPE0	RTR, FRAG, TCP	REED	Compute Bound	Real Time
TYPE1	RTR, FRAG, TCP	CAST	Compute Bound	Real Time
TYPE2	RTR, FRAG, TCP	CAST, REED	Compute Bound	Content-Delivery
TYPE3	RTR, FRAG, TCP	REED, JPEG	Data Bound	Content-Delivery
TYPE4	RTR, FRAG, TCP	CAST, REED, JPEG	Data Bound	Content-Delivery

The two broad categories of IP Packets are hard real-time termed as *real-time* packets and soft real-time termed as *content-delivery* packets. *Real-time* packets are assigned with high priority whereas *content-delivery* packets are processed with lower priorities. The total propagation delay (source to destination) of real-time packets is less than 150 milliseconds (ms) and less than 10 sec for content-delivery packets respectively.

Table 3-3: Performance Targets for IP packet type

Application/Packet Type	**Data Rate**	**Size**	**End-to-end Delay**	**Description**
Audio	4 – 64 (Kb/s)	< 1KB	< 150 msec	Conversational Audio
Video	16 – 384	~ 10KB	< 150 msec	Interactive video

	(Kb/s)			
Data	-	~ 10KB	< 250 msec	Bulk data
Still Image	-	< 100KB	< 10 sec	Images/Movie clips

Assuming maximum 10 to 15 hops are allowed per packet, worst case processing time of the packets in the intermediate routers is in the order of 10ms in case of real-time packets and 1000 ms in case of content-delivery packets respectively [52]. The network propagation delay is assumed to be negligible as optical fiber networks provide sufficient data bandwidth [8]. Table 3-3 enlists the end-to-end transmission delays associated with each packet categories. All of our candidate micro-architectures must be designed to process packets within the packet processing delay limits. In addition to processing delay per packet, we also consider total number of packets required to process per second in a TBPS router. Since IPv6 packets are of varying length we assume in average packet contains a payload of size 8KB. Hence, total number of packets to be processed is given by,

$$Packets\ per\ second = \frac{Bandwidth}{Average\ packet\ size} \qquad (3-1)$$

According to Equation 3-1 approximately 70 to 100 million packets are required to be processed per second to achieve TBPS line speed. In a shared memory NeP with N_C number of cores where each core has N_T hardware threads, $N_C * N_T$ packets are processed simultaneously.

Table 3-4: Processing time and instruction count of 5 packet types

Packet Type	**Processing Time (msec)**	**Instruction Count**	**Packet Distribution**
TYPE0	10	1255368	60%
TYPE1	10	1354559	25%
TYPE2	10	1258022	5%
TYPE3	1000	8922987	5%
TYPE4	1000	9124851	5%

The processing time, instruction count and packet distribution for all the packet types are enlisted in Table 3-4. For a given network bandwidth the total number of packets to be processed per second contains a distribution of different packet types. For example, if 100 packets are to be processed per second, packet distribution percentage as shown in Table 3-4 signifies that there are 60 TYPE0 packets, 25 TYPE1 packets and 5 packets of types TYPE2, TYPE3 and TYPE4.

CHAPTER 4: CASPER PROCESSOR SIMULATOR

CASPER is an instruction trace-driven cycle-accurate many-core processor simulator which models a shared memory heterogeneous architecture. CASPER provides the user with three key benefits – *(i)* entire SPARCV9 instruction set support enabling the user to run any Solaris executable on the simulator, *(ii)* a large set of tunable architectural parameters so that heterogeneous CMT design space can be widely explored, and *(iii) deep chip vision* - accurate area and performance estimations, along with cycle-accurate power and energy consumption models, which enable the user to capture energy consumption characteristics of different parts of the chip on a cycle-by-cycle basis. CASPER also provides the architect complete access to the processor and enables the monitoring of critical system events. CASPER is open-sourced under GNU GPL license [53].

CASPER is written in C++ programming language and has been flexibly parallelized using pthreads to optimally run on a wide variety of parallel processors. Functionally, it has been validated against the open-sourced functional simulator of Sun Microsystem's UltraSPARC T1 processor [54-56] - SPARC Architecture Simulator (SAM). Timing verification is done in two stages – *(i)* CPI and memory operations of applications executed on UltraSPARC T1 processor and a structurally similar design simulated in CASPER are matched

and *(ii)* number of retired instructions, required number of cycles to commit these instructions and program counter progression are matched with the pre-characterized HDL models of the processor.

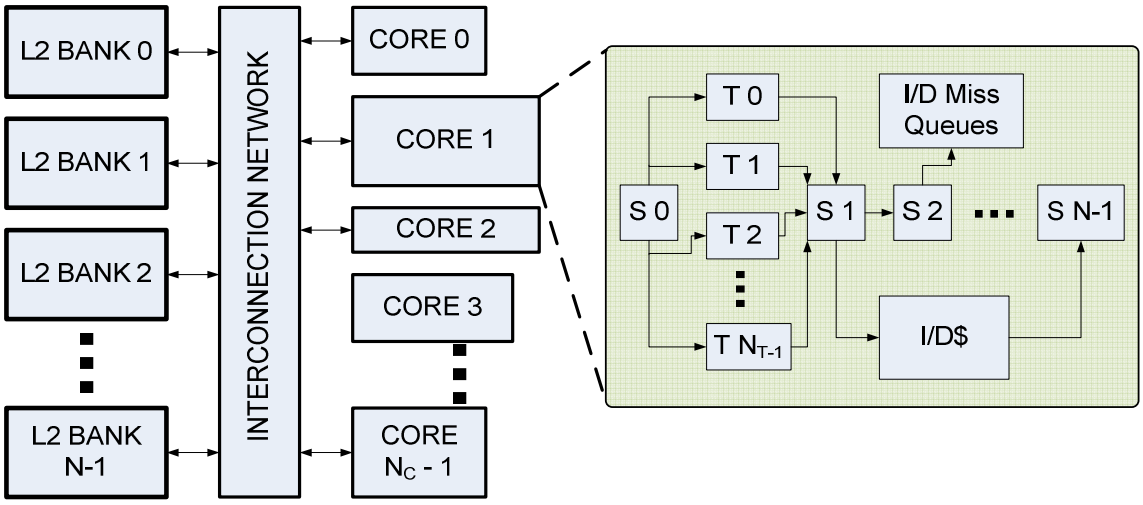

Figure 4-1: The shared memory processor model simulated in CASPER. N_C heterogeneous cores are connected to N_B banks of shared secondary cache via a crossbar interconnection network. Each core consists of S_0 to S_{N-1} are the pipeline stages, T_0 to T_{NT-1} hardware threads, L1 I/D cache and I/D miss queues

4.1 Processor Model

The processor model used in CASPER is shown in Figure 4-1. N_C cores are connected to the shared L2 cache through a crossbar interconnection network. The unified L2 cache is inclusive and is divided into N_B banks. Each bank of L2 privately owns DRAM controllers and independently communicates with the RAM modules. N_C and N_B are parameterized in CASPER. The 64-bit pipeline is parameterized to handle N_T hardware threads and is divided into 6 main stages – Instruction-Fetch (F-stage), Thread-Schedule (S-stage), Branch-and-Decode (D-stage), Execution (E-stage), Memory-Access (M-stage) and Write-back (W-stage).

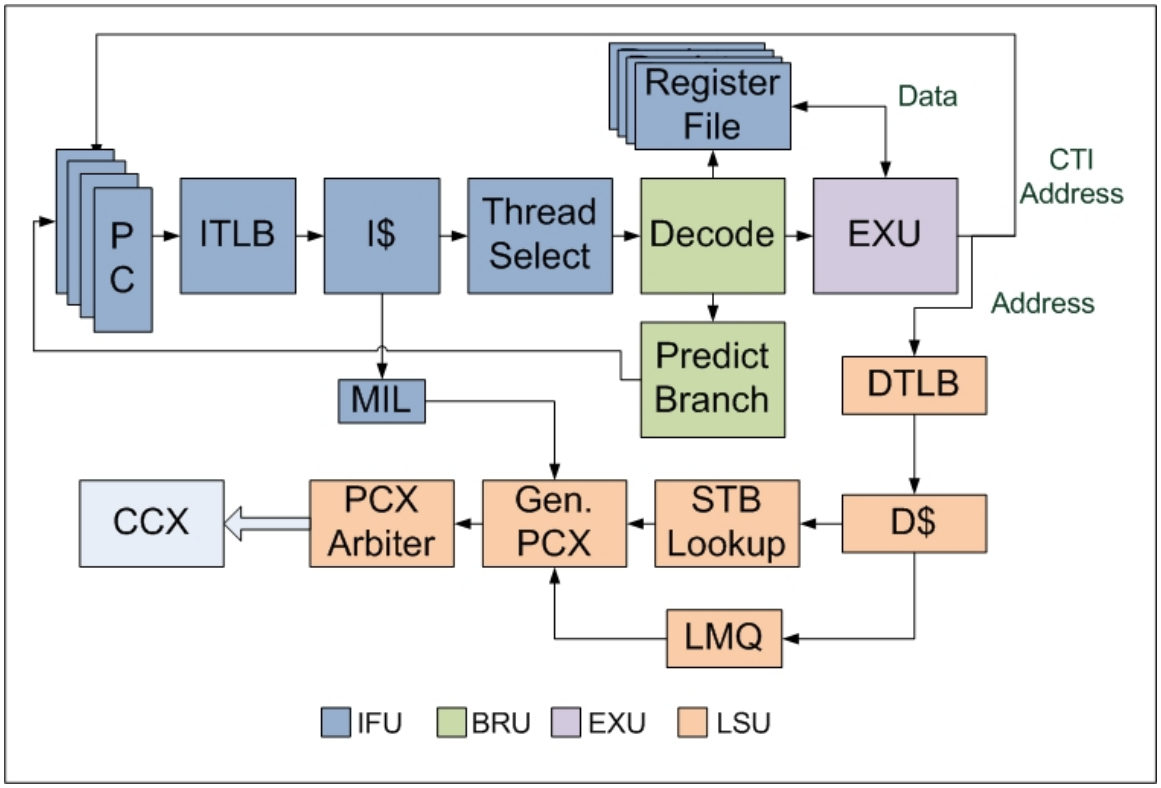

Figure 4-2: Micro-architectural structures inside a core in CASPER

Figure 4-2 shows the different stages of the in-order instruction pipeline inside a core. The Instruction Fetch Unit includes the instruction address translation buffer (I-TLB) and the instruction cache (I$) and the thread scheduling state machine. I-TLB and I$ are shared by the hardware threads. Each thread privately owns a register file (processor-state specific set of registers) and a set of alternate address mapped registers called ASI registers; the D-stage includes a full SPARCV9 instruction set decoder described in [57]. The E-stage includes a standard RISC 64-bit ALU, an integer multiplier and divider. Load Store Unit (LSU) is the top level module which implements the M-stage and W-stage. It also includes the data TLB (D-TLB) and data cache (D$).

The miss path of I$ is controlled through Missed Instruction List (MIL) and Instruction Fetch Queue (IFQ), while that of the D$ is controlled through Load Miss Queue (LMQ) which maintains cache misses separately for each thread. Duplicate load misses are maintained in a wait buffer to reduce off-core traffic. Store Buffer (SB) serializes all the stores following the Total Store Order (TSO) model.

The Floating point Unit (FPU) which executes single and double-precision floating-point operations can either be shared across all cores or can be privately owned by a single core. In the former case, all floating point operation packets are routed to the FPU via the interconnection network. Two thread scheduling schemes are implemented in CASPER. The *small latency thread scheduling* scheme allows instructions from ready threads to be issued into the D-stage at every clock cycle [56, 58]. *Long latency scheduling* scheme allows one active thread to continue its execution till it is complete or interrupted by higher priority threads. The full list of tunable architectural parameters is given in Table 4-1.

Table 4-1: Configurable Parameters in Casper

Name	Range	Description
Cores	1: N_C	Number of cores on chip
Strands	1:N_S	Hardware threads per core
Strand Scheduling	2	Long Latency Scheduling / Small Latency Scheduling
FPU	1 or 0	FPU can be shared between the cores or threads

Name	Range	Description
I$_C/D$_C	4:64 (KB)	Size of L1 I-D cache
I$_B/D$_B	4:64	Size L1 I-D cache block
I$_A/D$_A	2:8	Associativity of L1 I-D cache
I$/D$ Hit Latency	2:4 clock cycles	Measured in Cacti for 45nm technology
IFQ	$1N_S$:$8N_S$	Size of Instruction Fetch Queue
MIL	$1N_S$:$8N_S$	Size of Missed Instruction List
BBUFF	$4N_S$:$16N_S$ entries	Size of Branch Address Buffer
LMQ	$1N_S$:$8N_S$	Size of Load Miss Queue
DFQ	$1N_S$:$8N_S$	Size of Data Fill Queue
SB	$1N_S$:$16N_S$	Size of Store Buffer (Store-ordering)
L2$_C	256KB:16MB	Size of L2 cache
L2$_B	8:24	Size of L2 cache block
L2$_A	4:16	Associativity of L2 cache
L2$_NB	4:16	Number of L2 cache banks

In case of heterogeneous designs, the cores in CASPER are configured with different micro-architectures (one set of values of the architectural parameters) although the six functional stages of the core pipeline are fixed. The size and structure of the core-to-memory and memory-to-core request packets are also kept same across all the cores for simplicity. This is important since the

size of the interface packets usually depends on the cache block sizes. The clock signals to the heterogeneous cores are designed to be scaled so that different cores can be driven at different voltage and frequency levels. The tunable parameters in L2 cache are number of banks, bank size, associativity, block size and access latency. Arbiters in the L2 cache controllers issues one request packet from the input queues at a time.

4.2 Performance Measurement

For a given set of micro-architectural parameters, CASPER uses counters in each core to measure the number of completed instructions individually for each hardware thread ($Instr_{THREAD}$) and for the entire core ($Instr_{CORE}$) every second. For a processor clock frequency of 1GHz, the total number of clock cycles per second is 1G. In this case the CPI-per-core is calculated as ($1G/Instr_{CORE}$) while CPI-per-thread is calculated as ($1G/Instr_{THREAD}$).

In addition to CPI, counters are provided in CASPER to measure *(i)* pipeline stalls, *(ii)* wait time of threads due to MIL/LMQ/SB being full, *(iii)* I$ and D$ misses, and *(iv)* stalls due to other long latency operations such as ASI registers writes and floating point operations. Counters are also attached to the crossbar network to measure the access frequencies of the various cores and threads in them. The input queues of the L2 cache are monitored to track the accesses occurring every clock cycle from the various cores and corresponding threads. In addition, special counters are attached to every *set* in the L2 cache to report utilization, number of *hits*/*misses* per core and per hardware thread, and *reuse* and *access* frequencies of the active threads running in the system [59]. Cache hit

latencies (delays) are measured using Cacti [60, 61] for a given cache size, block size, associativity and silicon technology. Miss penalties are counted in clock cycles by the counters provided in CASPER.

Another important feature used in CASPER is *Hardware Scouting.* Usually long latency operations such as ASI register load/stores, I$ misses and D$ load misses in an in-order thread are blocking in nature. This means the blocked thread is in a WAIT state and no further instructions are issued into the decode stage. This also means that even though the depth of the load miss queue (LMQ) is greater than one, only one entry is effectively used. To save a few more clock cycles such that load misses following a previous load misses are also enqueued in the LMQ, hardware scouting is implemented in our pipeline which switches the state of a blocked thread to SPECULATIVE RUN state instead of WAIT state. Instructions in a thread which is in SPECULATIVE RUN state are scheduled to the decode stage, but are never committed until the first blocking load miss is resolved. Once the first load miss is resolved, the thread is switched to usual READY state and further execution continues. Arithmetic instructions appearing between two load misses are rolled back and the issuing thread is kept waiting till the first load miss is committed. In average, this enhances the performance of a single thread by 2-5%.

4.3 Verification

Functional correctness of candidate designs simulated in CASPER is verified using a set of diagnostic codes which are designed to test all the possible instruction and data paths in the stages of the pipeline in a core. Additional set of

diagnostic codes are written which consist of random combinations of instructions such that different system events such as traps, store buffer full and others are also asserted. To further verify the accuracy of CASPER, we have compared the total number of system events generated while executing 10 IP packets in the ENePBench in a real-life UltraSPARC T1000 machine consisting of an UltraSPARC T1 (T1) processor (T1) [56] to an exact UltraSPARC T1 prototype (T1_V) simulated in CASPER. UltraSPARC T1 is the closest in-order CMT variant to our CMT designs modeled in CASPER and consists of 8 cores and 4 hardware threads per core. The simulated processor in CASPER had equal number of cores, hardware threads per core, L1 and L2 caches as T1. Columns 3a, 3b, 4a, 4b, 5a, 5b and 6 of our results tabulated in Table 4-2 compare the number of instructions committed, store buffer full event, I$ misses and D$ misses respectively in T1 and T1_V respectively. Column 6 shows that in average, the error in number of system events is less than 10%.

Table 4-2: Comparison between number of system events for 5 IP packets types in (i) T1000 server with an UltraSPARC T1 processor and (ii) a T1 prototype simulated in CASPER

Packet Type	**Clock Ticks (in 10^6)**	**Instr_cnt (in 10^6)**		**SB_full (in 10^3)**		**IC_misses (in 10^3)**		**DC_misses (in 10^3)**		**Avg. Error (%)**
		T1	**T1_V**	**T1**	**T1_V**	**T1**	**T1_V**	**T1**	**T1_V**	
TYPE0	0.674	0.255	0.255	5.0	4.9	2.6	2.6	1.56	1.59	2.01
TYPE1	0.673	0.254	0.254	5.4	5.6	2.5	2.4	1.50	1.6	7.35

TYPE2	0.612	0.26	0.258	5.1	5.2	2.6	2.5	1.51	1.52	4.0
TYPE3	2.257	0.90	0.892	12.9	12.7	3.5	3.9	6.84	6.84	5.7
TYPE4	2.259	0.94	0.896	18.9	17.1	3.5	3.6	6.89	6.89	9.5

4.4 Deep Chip Vision – Area and Power/Energy Measurement

To accurately model the area and the power dissipation of the architectural components we *(i)* design scalable hardware models of all pipelined and non-pipelined components of the processor in terms of corresponding architectural parameters (Table 1), *(ii)* derive area and power dissipations (dynamic + leakage) of the component HDL models using industry-standard synthesis and layout tools such as Synopsys and placement and routing tools as Encounter and *(iii)* statistically curve-fit the area and power dissipation values of the components for increasing values of the parameter to derive linear estimation models. Derived power models are then used to estimate energy consumption of the components by capturing the *activity factor* $\alpha(t)$ from simulation, and integrating the product of power dissipation and $\alpha(t)$ over simulation time.

4.5 Design of HDL Models

Table 4-3 summarizes the common hardware structural components used in a CMT processor and the HDL models they map to. Some of the HDL models of the components (both intra-core and chip level components such as interconnect buses and arbiters) are available in OpenSPARC [56], while others have been custom designed in our lab. The HDL models are designed to be scalable, and capture different variations in the architectural parameters.

Table 4-3: Common CPU Hardware Structures and their models used in CASPER

Hardware Structure	**HDL Model**	**Affected By**
I$, D$	Cache Array	
Branch Predictor	RAM + Logic	threads-per-core (N_S)
I-TLB, D-TLB	RAM + CAM	-
Load Miss Queue	RAM + CAM	N_S
Missed Instruction List	RAM + CAM	N_S
Store Buffer	RAM + CAM + Logic	N_S
Crossbar Interconnect	Scaled CCX	number of cores (N_C)
L2 Cache Banks	Cache Array	N_C
FPU	Logic	SPARCV9 Floating Point Operations – FADD, FSUB, FMUL, FDIV
Integer + Float Register File	Logic	SPARCV9 Register File

4.5.1 Area and Power Estimation

To accurately model the area and the power dissipation of the architectural

components we have *(i)* designed scalable hardware models of all pipelined and non-pipelined components of the processor in terms of corresponding architectural parameters, *(ii)* derived power dissipations (dynamic + leakage) of the component HDL models using industry-standard synthesis and layout tools such as Synopsys [62] which targets the Berkeley 45nm Predictive Technology Model (PTM) technology library [63] and placement and routing tools as Encounter [64] and *(iii)* statistically curve-fit the area and power dissipation values of the components for increasing values of the parameter to derive linear regression models. Derived power models are then used to estimate energy consumption of the components by capturing the *activity factor* $\alpha(t)$ from simulation, and integrating the product of power dissipation and $\alpha(t)$ over simulation time. The following equation is used to calculate the power dissipation of a pipeline stage –

$$P_{stage}(t) = P_{leakage}(t) + \alpha P_{dynamic}(t) \quad (4-1)$$

where $\propto$ is the *activity factor* of that stage (α=1 if that stage is *active*; α = 0 otherwise) which is reported by CASPER, and $P_{leakage}$ and $P_{dynamic}$ are the leakage and dynamic power dissipations of the stage respectively.

The power dissipation values of the parameterized micro-architectural non-pipeline components in a core namely, the load miss queue, store buffer, missed instruction list, I/D-TLB, and I/D$ are collected from Cadence Encounter using a 1GHz clock into lookup tables. These lookup tables are then used in the simulation to calculate the power dissipation cycle by cycle. Table 4-4 shows the area, dynamic and leakage powers of the micro-architectural blocks in a core.

Area, delay and power dissipation of caches in Table 4-4 have been modeled using Cacti 4.1 [60, 61].

Table 4-4: Post-Layout Area, Dynamic and Leakage Power of HDL Models

HDL Model	**Area (mm^2)**	**Dynamic Power (mW)**	**Leakage Power (uW)**
RAM (16)	0.022	1.03	17.81
CAM (16)	0.066	3.51	67.70
FIFO (16) for 8 threads	0.3954	165	1200.00
TLB (64)	0.0178	21.11	92.60
Cache (32KB)	0.0149	28.3	-
Cache Controller			
Integer Register File	0.5367	11.92	4913.7
Float Register File	0.0764	309.44	551.897
FPU	-	-	-
IFU	0.0451	3280.1	378.39
EXU	0.0307	786.99	301.94
LSU	0.8712	5495.3	6848.30
TLU	0.064	1302.2	553.8458
Floating Frontend Unit (FFU)	0.0123	767.07	98.40
Multiplier	0.0324	23.74	383.88

The area distribution of a 8 threaded core in CASPER is shown in Figure 4-3. The width of LMQ, SB, D-TLB are 16, 16 and 64 respectively. The D$ is 32KB in size with 16B linesize and 4 set-associativity. The integer register file (IRF) for each thread is segregated and hence there are 8 in total. IRF is the biggest contributor of the core area.

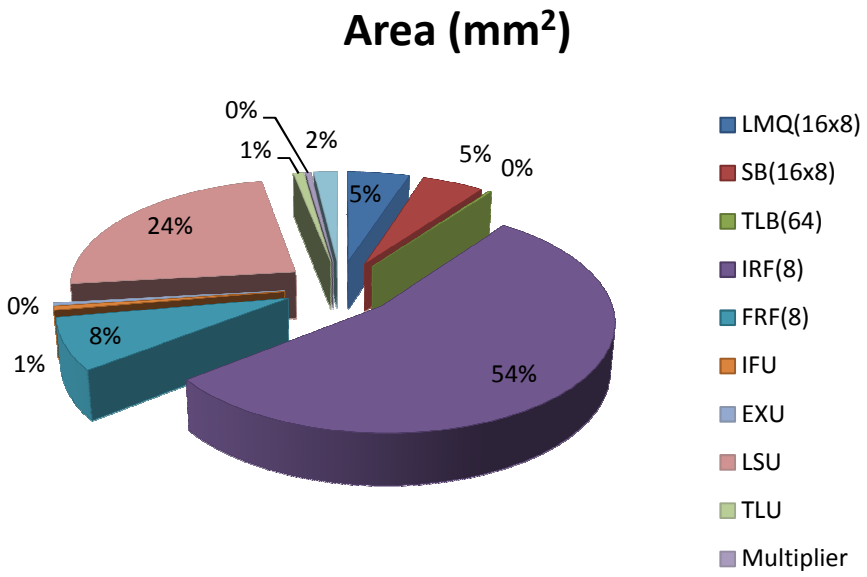

Figure 4-3: Area distribution of the micro-architectural features of an 8-thread single core area estimation model in CASPER

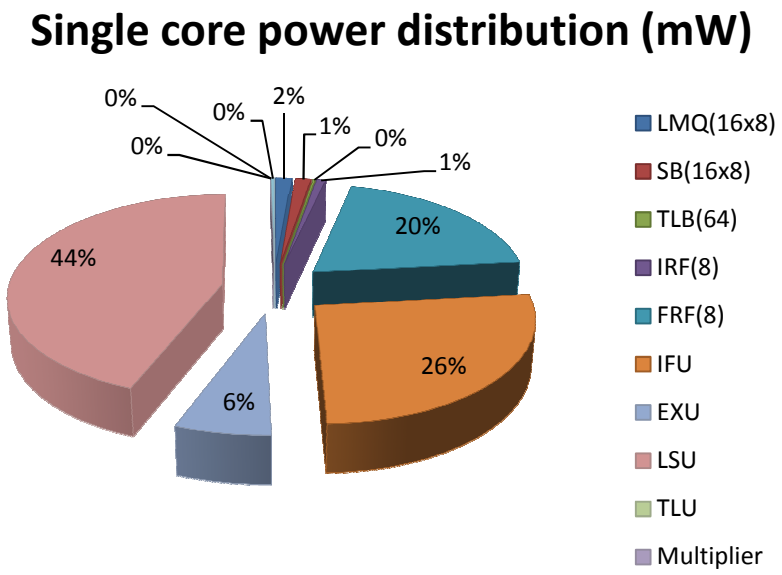

Figure 4-4: Power dissipation distribution of the micro-architectural features of an 8-thread single core area estimation model in CASPER

Figure 4-4 shows the power estimation distribution of the micro-architectural blocks of an 8-threaded core simulated for 1 million clock cycles in CASPER. The width of LMQ, SB, D-TLB are 16, 16 and 64 respectively. The D$ is 32KB in size with 16B linesize and 4 set-associativity. As observed the load store unit or LSU is the heaviest contributor of power dissipation in a core. The operations of LSU is divided into four pipelined stages where the D-TLB, D$ are accessed, Store Buffer (SB) is cammed and all out-going packets are resolved; hence the high power dissipation.

The structures of shared L2 cache and interconnection network is co-dependent on the number of cores in a candidate design which makes it immensely difficult to synthesize, place and route all possible combinations. Hence we have used *multiple linear regression* [65] to model the throughput, dynamic and leakage power dissipation of shared L2 cache and interconnection

network respectively. A detailed discussion about the multiple linear and non-linear regressions method is presented in Chapter 6. Dynamic power dissipation measured in Watts of L2 cache is related to the size in megabytes, associativity and number of banks N_B as shown in Equation 4-2. The model parameters are shown in Table 4-5.

$$L2_{dynamic\_power} = c0 + c1 * Size + c2 * Associativity - c3 * N_B \quad (4-2)$$

Table 4-5: L2 cache linear regresssion model parameters

R	R Square	Std. Error of Estimate
0.926	0.857	0.524

Similarly, the dynamic and leakage power dissipation measure din milli-Watts of a crossbar interconnection network is given by Equation 4-3 and Equation 4-4 respectively. Note that dynamic and leakage power is exponentially related to number of cores (N_C) and number of cache banks (N_B). In these two cases, the R value is 0.753 and standard errors of estimates is10.64.

$$IN_{dynamic\_power} = b0 + b1 * e^{b2*N_C} + b3 * Q + b4 * e^{b5*N_B} \quad (4-3)$$

$$IN_{leakage\_power} = b0 + b1 * e^{b2*NC+b3*Q+b4*NB} \quad (4-4)$$

4.5.2 Modeling Activity Factor

It is necessary to track the activity factors of all the components and all the stages to accurately estimate the energy consumption of a design. Cycle-accurate simulation captures the switching activity of the micro-architectural components in every clock cycle. As a given instruction is executed through the multiple stages

of the instruction pipeline inside a core, the simulator tracks *(i)* the intra-core components that are actively involved in the execution of that instruction and *(ii)* the cycles during which that instruction uses any particular pipeline stage of a given component. Any component or a stage inside a component is assumed to be in two states – *idle* (not involved in the execution of an instruction) and *active* (process an instruction). For example, in case of a D$ load-miss, the occurrence of the miss will be identified in the M-stage. The load instruction will then be added to the LMQ and W-stage will be set to an *idle* state for the next clock cycle.

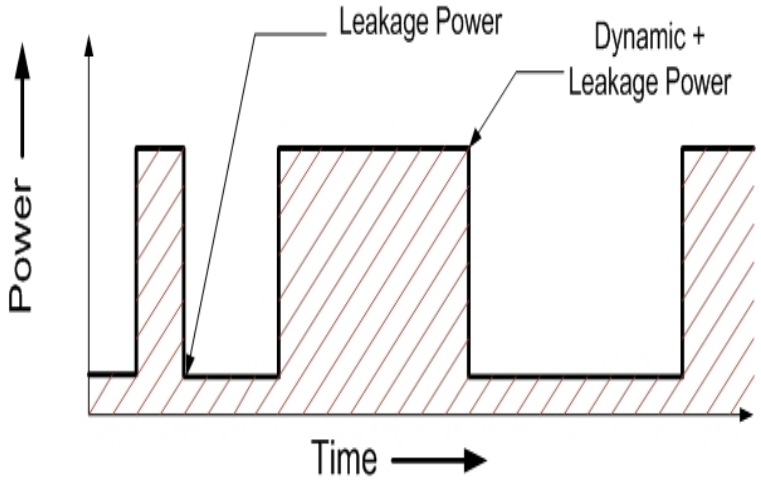

Figure 4-5: Power Dissipation transient for a single pipeline stage in a component. The area under the curve is the total Energy consumption

A non-pipelined component is treated as a special case of a single stage pipelined one. We consider only leakage power dissipation in the *idle* state and both leakage and dynamic power dissipations in the active state. Figure 4-5 shows the total power dissipation of a single representative pipeline stage in a component. Note that the total power reduces to just the leakage part in the

absence of a valid instruction in that stage (*idle*), and the average dynamic power of the stage is added when an instruction is processed (*active*).

A certain pipeline stage of a component will switch to *active* state when it receives an *instruction ready* signal from its previous stage. In the absence of the *instruction ready* signal, the stage switches back to *idle* state. Note that the *instruction ready* signal is used to clock-gate (disable the clock to all logic of) an entire component or a single pipeline stage inside the component to save dynamic power. Hence we only consider leakage power dissipation in the absence of an active instruction. In case of an instruction waiting for memory access or in the stall state due to a prior long latency operation, is assumed to be in *active* state.

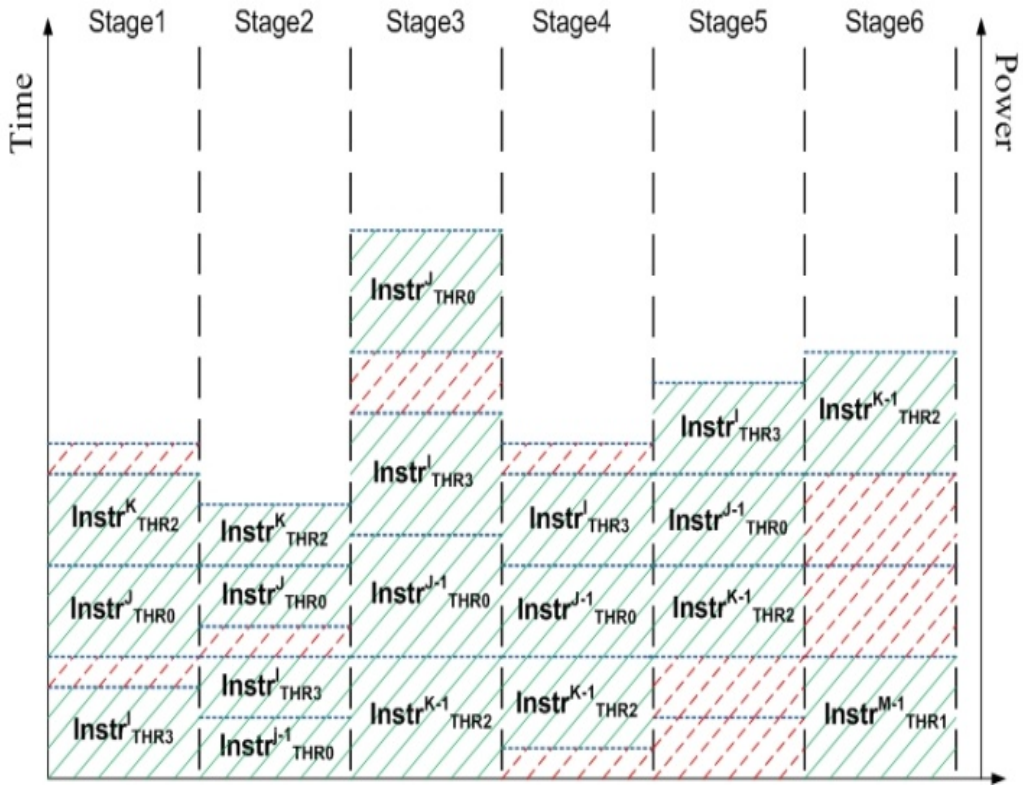

Figure 4-6: Power profile of a pipelined component where multiple instructions exist in different stages. Dotted parts of the pipeline are in idle state and add to

the leakage power dissipation. Shaded parts of the pipeline are active and contribute towards both dynamic and leakage dissipate power dissipations.

Figure 4-6 shows the power (dynamic + leakage) contributions of multiple pipeline stages (which are simultaneously processing different pipelined instructions) to generate the total power dissipation profile of a pipelined micro-architectural component. As shown in the Figure 4-6 above, for any given pipeline stage the horizontal separation lines correspond to different clock cycles during which different instructions flow through the stage. The shaded parts correspond to *active* states of the stage (dynamic + leakage power), while the dotted parts correspond to *idle* states of the stage (only leakage power). Note that different stages have different values of dynamic and leakage power dissipations. The following equation is used to calculate the power dissipation of a pipeline stage –

$$P_{stage}(t) = P_{leakage}(t) + \alpha P_{dynamic}(t) \qquad (4-5)$$

where $\propto$ is the *activity factor* of that stage (α=1 if that stage is *active*; α = 0 otherwise) which is reported by CASPER, and $P_{leakage}$ and $P_{dynamic}$ are the leakage and dynamic power dissipations of the stage respectively. Finally the energy consumption is found using the following equation:

$$E = \sum_{t=1}^{Simulation\ Time} P_{stage}(t) \qquad (4-6)$$

A trace of the total power dissipation of a processor under simulation is reported by CASPER by adding the power dissipation profiles of all stages of all components for every clock cycle of simulation. The area under that curve is the total energy consumption of the processor for a given benchmark. CASPER can be used to design future throughput intensive CMT architectures ranging from

real-time constrained embedded systems (such as embedded network processors) to high-performance computing (HPC) platforms (such as web/application servers). Typically, for embedded applications the objective is to minimize energy consumption subject to throughput constraints, while for high-performance applications throughput is maximized under power dissipation constraints. The following sections explain the data demand characteristics of these two application domains and how we employ CASPER to design processors for them. We use commercial benchmarks such as CommBench-0.5 (embedded network processors) and SPECJBB2005 (web/application server processors) to evaluate CMT architectures.

4.5.3 Design Trade-offs in case of SPECWEB2005

High-end web and applications servers process huge amounts of data simultaneously. The typical data complexity is of the order of 10-20 million simultaneous users (parallel tasks) accessing large databases and executing transactions. User processes are mapped to software threads and corresponding transaction data from the backend database servers are transferred into the local memory of the executing processor. It is possible that substantial data can be reused or shared for multiple users justifying the use of shared memory architectures to enhance performance. CMT shared-memory architectures are known to perform efficiently for such applications [54, 55, 58]. L2 cache size (maximize amount of on-chip data), number of threads per core, number of cores and other critical architectural parameters have substantial impact on processor performance. Table 4-6 lists the parameters of interest and the range of values

that can be explored in an HPC CMT processor. The benchmark used to study this class of applications is SPECJBB2005 [66].

Table 4-6: The micro-architectural parameters and their ranges used to study the design trade-offs in SPECWEB2005

Parameter	Range
Cores	1:32
Threads	4:32
L1_I$	4KB:64KB
L1_D$	4KB:64KB
I-TLB	16:64
D-TLB	16:64
L2$_Size	1MB:16MB
L2$_banks	4:16

Table 4-7: Baseline Architecture used to measure CPI for SPECJBB2005

Parameter	Value
Cores	1
Threads per core	1
L1 I$/D$ size	16KB/8KB
L1 I$/D$ Associativity	4/4
L1 I$/D$ Block size	32/16
I/D-TLB	64/64

Table 4-7 shows the baseline architecture used to study the SPECJBB2005 benchmark. Figure 4-7 shows how the normalized CPI of a core and that of a strand vary with the number of threads in the core (from 1 to 16). Other architectural parameters are kept constant in baseline architecture for this experiment. As observed, performance of a core levels off as the number of threads increases, leading to more L1 cache misses due to cache thrashing.

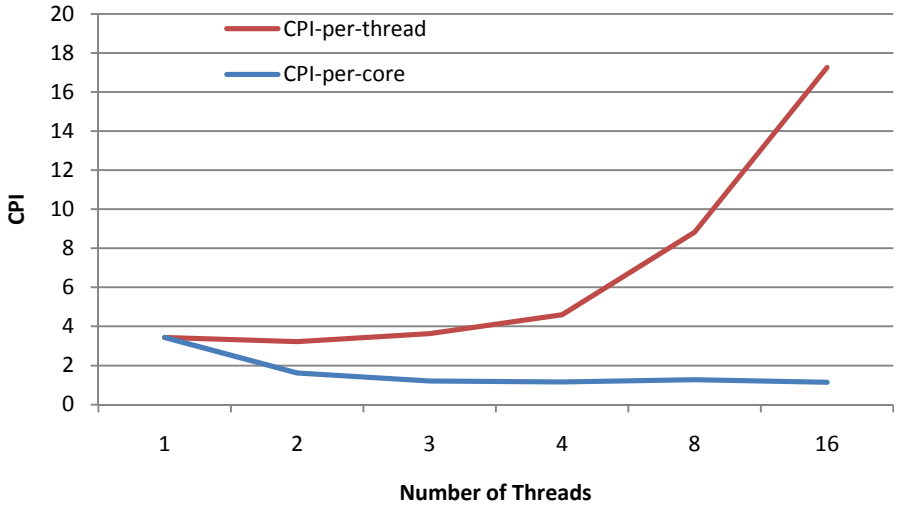

Figure 4-7: Scalability of CPI-per-core and CPI-per-strand of a core as threads-per-core is increased from 1 to 16. SPECJBB2005 is used as benchmark.

Additional stalls occur in the LSU, where the arbiter, responsible for transferring load-store packets from the core to the shared L2 cache becomes the bottleneck. The arbiter follows a round-robin fairness scheduling scheme to issue packets into the interconnection network. The worst case wait time for an outbound load/store packet of a thread is of the order of $O(N_S)$, where N_S is the number of threads in the core. However, the CPI-per-strand increases considerably since a thread can wait in the worst case for $O(N_S)$ cycles to be scheduled from the F-

stage to the D-stage. Note that Figure 6 shows the data for *small latency thread scheduling scheme*.

In Figure 4-8, we show how performance scales when two independent architectural parameters, *(i)* number of threads per core and *(ii)* size of L1 data cache, are varied together. The area and power dissipation for a 4-way set associative 16-block size cache for increasing size of cache is enlisted in Table A.1 in Appendix A. Note in Figure 4-8 that less misses in larger caches helps in increasing the effective CPI-per-core. However, as the number of threads-per-core increases, L1 cache contention reduces this benefit. However, larger cache sizes indicate larger die area and hence more power dissipation.

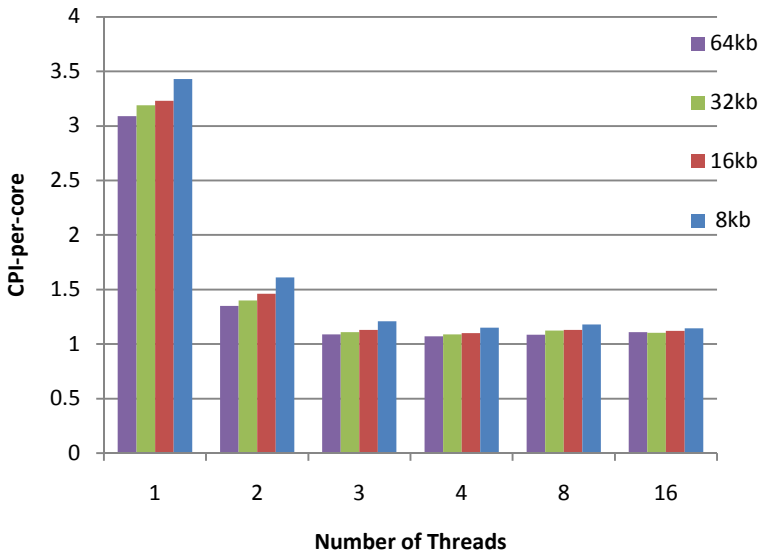

Figure 4-8: CPI-per-core scalability as threads-per-core is scaled from 1 to 16 and the size of a 4-way set associative 16-block sized data cache is varied from 8-64KB (with SPECJBB2005)

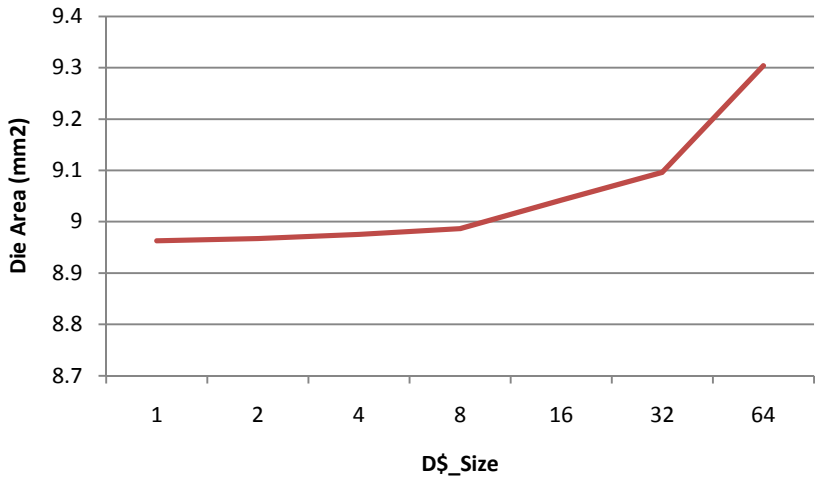

Figure 4-9: Scalability of the area of a core (consisting of 4-threads) as size of a 4-way set-associative 16-block size data cache increases (with SPECJBB2005)

Figure 4-9 shows die area (in mm^2) of one core scales almost super-linearly with increasing cache sizes. In the figure we vary the data cache size from 1KB to 64KB.

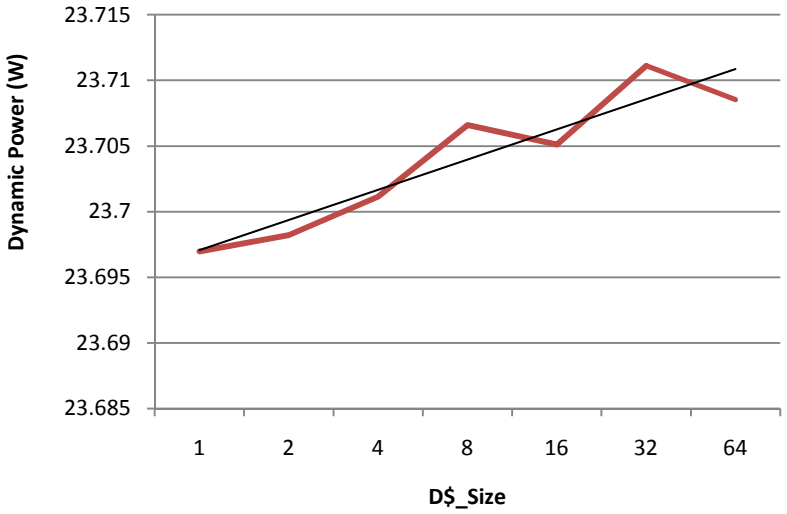

Figure 4-10: Dynamic power dissipation of a core (consisting of 4-threads) as size of a 4-way set-associative 16-block size data cache increases. The black line shows the trend (with SPECJBB2005)

Figure 4-10 shows peak dynamic power dissipation of a core consisting of 4 threads as the size of the D$ varies from 1KB to 64KB. As expected with bigger cache sizes, we observe more power dissipation. A 32KB data cache size has high power dissipation ratings than that of 64KB cache as can be seen from Table A.1 in Appendix A. Also, the power saved due to lower cache misses for a 32KB data cache, compared to a 16KB data cache, is mitigated by the high power signature of the cache. This explains the slight increase in overall dynamic power dissipation of the core for this cache size.

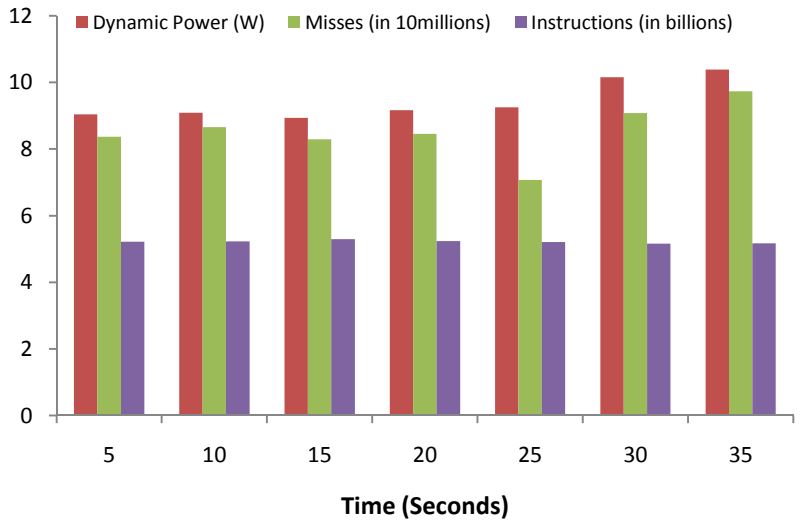

Figure 4-11: Dynamic Power Dissipation in a 4-threaded core simulated in CASPER with SPECJBB2005. Number of retired instructions at each time step is around 5.2 billion.

Figure 4-11 shows the variation in power dissipation of a 4-threaded core according to data cache misses and committed instructions. Note the close correlation between D$ misses and dynamic power dissipation.

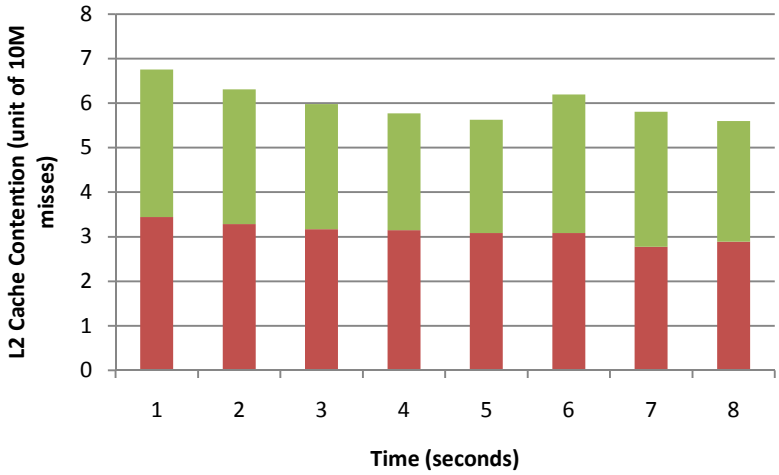

Figure 4-12: Shared L2 cache contention as a function of time for a 2-core CMT (4 threads-per-core) processor (with SPECJBB2005); L1 D$ misses of Core_0 and Core_1 are shown in red and green respectively.

Figure 4-12 shows traffic in the shared L2 cache (in units of 10 million misses) due to the simultaneous accesses from two cores in the system. In this case, the cores are configured with 4 threads-per-core, a 8KB data cache, and other features similar to the baseline architecture described in Table 4-7. Table 4-8 shows average L2 access load (in units of 10 million misses) as the number of cores is scaled, using similar core configurations as described above.

Table 4-8: Average L2$ load distribution as the number of 4-threaded cores is increased from 2 to 8. Data corresponds to units of 10 million misses

Cores	**C2T4**	**C4T4**	**C8T4**
Core0	3.11	3.19	3.24
Core1	2.89	3.12	3.27
Core2	0	3.24	3.26
Core3	0	3.098	3.33

Core4	0	0	3.3
Core5	0	0	3.26
Core6	0	0	3.21
Core7	0	0	3.33

4.5.4 Design Trade-offs in case of EnePBench

NePs must process data packets at line speeds of typically 50-60 million packets per second. NePs execute codes for all layers of the Open System Interconnection (OSI) Protocol Stack [67]. The operations performed on a typical example packet have been discussed in [19]. Usually, a packet is mapped to a software-thread (posix thread) where the functions from different layers are executed sequentially. In a many-core processor, different operations on a packet will be mapped to one of the hardware threads. In addition latency, due to stalls in packet processing due to dependency on other packets or other network state information, can be potentially hidden by overlapping multi-threaded execution.

Table 4-9: Architecture Parameters for Real-Time Embedded Network Processing

Parameter	**Range**
Cores	1:4
Threads	1:8
L1 I$	1KB:4KB
L1 D$	1KB:4KB
I/D-TLB	4:8

A subset of micro-architectural design parameters and their ranges are listed in Table 4-9. The benchmark suite used for these experiments is CommBench-0.5 [51] which is designed to measure performance of embedded NPs. The applications in CommBench are broadly categorized into *(i)* Header – Processing Applications (HPA) and *(ii)* Payload-Processing Applications (PPA). HPA programs include the following:

1. **RTR** - A Radix-Tree routing table lookup program.
2. **FRAG** - An IP packet fragmentation code.
3. **DRR** - Deficit Round Robin fair scheduling algorithm.
4. **TCP** - A traffic monitoring application.

PPA applications include:

a) **CAST** - A 128 bit block cipher algorithm.

b) **ZIP** - A data compression program based on commonly used Lempel-Ziv compression algorithm.

c) **REED** - An implementation of Reed-Solomon Forward Error Correction scheme.

d) **JPEG** - A lossy image data compression algorithm.

Table 4-10: Baseline Architecture used to measure CPI for CommBench 0.5

Parameter	Value
Cores	1
Threads per core	1
L1 I$/D$ size	1KB/1KB
L1 I$/D$ Associativity	2/2

L1 I$/D$ Block size	8/8
I/D-TLB	4/4

Table 4-10 shows the baseline architecture used. Table 4-11 shows the individual performance of the CommBench benchmark applications on the baseline architecture. Table 4-12 shows the energy consumption of the different CommBench programs. Column 2 shows the average energy consumption (with just clock-gating enabled in every stage in every component) – the LPMU algorithm described in Chapter 4 is not used. The data in column 3 shows the reduction in energy achieved by only power gating the core components. Power-gating reduces overall power dissipation by cutting down the leakage power. Due to relatively low leakage power in the 1-core 1-thread design, the effects of power-gating are relatively low. DVFS on the other hand produces larger reduction in energy consumption, as illustrated in the last two columns. In column 4, the supply voltage has been scaled down to 0.65V (from 0.7V), and the clock frequency has been to scale down to 0.8GHZ (compared to 1GHz for normal execution). Column 5 shows the energy reduction because of only frequency scaling (FS). The operating frequency for this experiment is set to 0.7GHz.

Table 4-11: List of system events for CommBench 0.5 applications

Benchmark	**Instruction Count**	**I$ misses**	**D$ misses**	**CPI**
RTR	1976779411	393164	725377	3.03
FRAG	1825056567	46150	420580	3.29

DRR	1776752836	103189	2837196	3.38
TCP	1829361746	20387	831234	3.28
CAST	1891153306	176163	1120180	3.18
ZIP	1834895294	31505	11679186	3.27
REED	4307088095	130365	583296	2.79
JPEG	1832782256	49717	4606153	3.28

Table 4-12: Power Dissipation for CommBench 0.5 applications with power-saving features

Benchmark	**Energy Consumption (µJ) for 8000 clock cycles**			
	without LPMU	**with power gating**	**with DVFS**	**with FS**
RTR	57.46	57.34	39.64	40.22
FRAG	55.59	55.27	38.35	38.90
DRR	57.34	56.48	39.55	40.14
TCP	57.91	57.78	39.95	40.54
CAST	57.20	57.10	39.46	40.04
ZIP	63.39	63.23	43.73	44.37
REED	65.63	65.08	45.27	45.94
JPEG	63.21	63.09	43.60	44.24

CHAPTER 5: DYNAMIC POWER MANAGEMENT TECHNIQUES IN CASPER

5.1 Abstract

Dynamic power management (DPM) in many-core processors executing parallel tasks involves a set of techniques which perform power-efficient computations under real-time constraints to achieve system throughput goals while minimizing power. DPM is executed by an integrated chip-wide power management unit (PMU), implemented in software, hardware or a combination thereof, which monitors and manages the power and performance of each core by dynamically adjusting its operating voltage and frequency. Hardware-controlled power management eliminates the computation overhead of the processor for workload performance and power estimations. Hence, hardware power management realizes more accurate and real time impact on workload performance than a slower reacting software power management can achieve. We evaluate three different hardware-controlled global PMU policies – (i) the existing chip-wide DVFS [68] and (ii) MaxBIPS [68] methods, besides (iii) the novel SmartBIPS algorithm that we have developed in this work. SmartBIPS uses a hysteresis based prediction mechanism for dynamic performance estimations, and thereby automatically incorporates shared memory interactions between the multiple cores. Results show that on average, SmartBIPS achieves a 41.3% improvement in power savings and a 19.8% improvement in throughput per unit

power with respect to MaxBIPS. This analysis is obtained using CASPER [69] running ENePBench the network packet processing benchmark discussed in Chapter 3. The throughput improvement of SmartBIPS with respect to chip-wide DVFS is 16.7% at a cost of 1.2 times higher power dissipation. MaxBIPS achieves a 61% throughput improvement at a cost of 2.1 times higher power with respect to chip-wide DVFS.

5.2 Introduction

Computing with power efficiency has become the paramount concern in embedded many-core platforms. High power dissipations in embedded platforms will increase form factors, reduce battery life, add to operation costs in cooling systems, and decrease the system reliability. Such concerns motivate the need for advanced power management schemes in embedded multi-core processors.

Dynamic power management (DPM) in many-core processors involves a set of techniques which perform power-efficient computations under real-time constraints to achieve system throughput goals while minimizing power. DPM is executed by an integrated chip-wide power management unit (PMU), which is typically implemented in software, hardware or a combination thereof. The PMU monitors and manages the power and performance of each core by dynamically adjusting its operating voltage and frequency. Power management is typically done using hierarchical power management units; the local power management unit (LPMU) optimizes power inside the core using techniques such as clock-gating [70] and power-gating [71], while the global power management unit (GPMU) at the chip level optimizes power dissipation using techniques such as

core-level dynamic voltage and frequency scaling (DVFS) [72]. Hardware-controlled power management eliminates the computation overhead that the processor incurs for software based power management while performing workload performance and power estimations. Hence hardware power management realizes more accurate and real time impact on workload performance than slower reacting software power management can achieve.

5.3 Dynamic Voltage and Frequency Scaling (DVFS)

The key idea of DVFS is to scale the voltages and frequencies of a single core or the entire processor during run-time to achieve specific throughputs while minimizing power dissipation, or to maximize throughput under a power budget. Equation 5-1 shows the quadratic and linear dependences of dynamic or switching power dissipation on the supply voltage and frequency respectively:

$$P = \alpha C V_{dd}^2 f \quad (5-1)$$

where α is the switching probability, C is the total transistor gate (or sink) capacitance of the entire module, V_{dd} is the supply voltage, and *f* is the clock frequency. Note that the system frequency needs to scale along with the voltage to satisfy the timing constraints of the circuit whose delay changes linearly with the operating voltage [68]. DVFS algorithms can be implemented at different levels such as the processor micro-architecture (hardware), the operating system scheduler, or the compiler [73]. Figure 5-1 shows a conceptual diagram implementing DVFS on a multi-core processor. Darker shaded regions represent cores operating at high voltage, while lighter shaded regions represent cores operating at low voltage. The unshaded cores are in sleep mode.

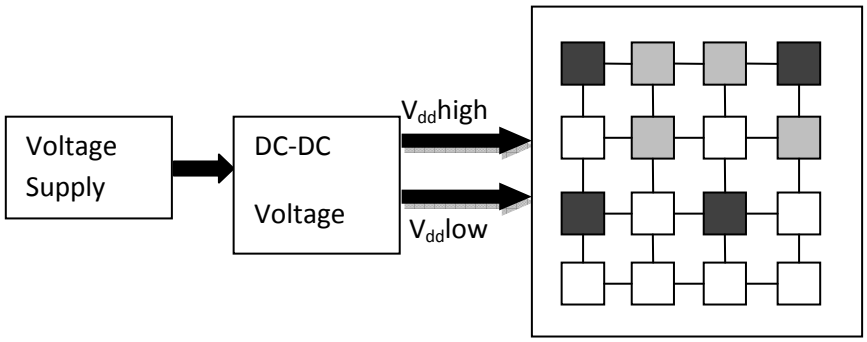

Figure 5-1: Dynamic voltage and frequency scaling for a multi-core processor

Until recently, the benefit of DVFS has been offset by slow off-chip voltage regulators that lack the ability to switch to different voltages in short time periods. This drawback motivates the need for fast on-chip DVFS control at the core level.

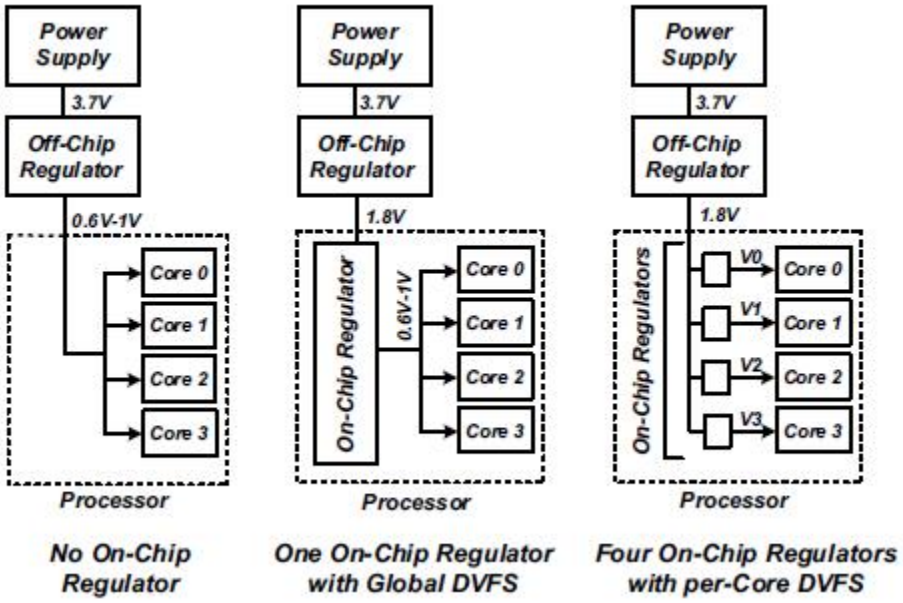

Figure 5-2: Three power-supply configurations for a 4-core CMP ***[74]***

Recent development of on-chip regulators with multiple on-chip power domains [74] has realized voltage regulation times of the order of 10mV per nanosecond. Figure 5-2 shows three possible power-supply configurations [74].

5.4 Hardware Controlled DPM in Commercial Embedded Processors

Some examples of commercial embedded processors which implement the DPM scheme include the Transmeta Crusoe, Intel StrongARM and XScale processors, and IBM Power4 [68]. These processors allow dynamically turning off idle sections of the processor, setting chip-wide fixed power consumption, halting idle cores, and/or operating dynamic voltage and frequency scaling of the cores in support of DPM strategies. Another commercial processor which partially implements the DPM scheme is the Intel Centrino Core Duo [75], which was designed to achieve two main goals: (1) maximize the performance under the thermal limitation the platform allows, (2) improve the battery life of the system relative to previous generations of processors. The OS views the Intel Core Duo processor as two independent execution parts, and the platform views the whole processor as a single entity for all power management related activities. Intel chose to separate the power management for a core from that of the full CPU and platform. This was achieved by making the power and thermal control unit part of the core logic and not part of the chipset as before. Migration of the power and thermal management flow into the processor allows the use of a hardware coordination mechanism in which each core can request any power saving state it wishes, thus allowing for individual core savings to be maximized.

5.5 Our Contribution

In this dissertation, we have developed a prototype of a new hardware-controlled power management algorithm called SmartBIPS, for multi-core processors with shared global resources such as hierarchical memory.

SmartBIPS is a DVFS based GPMU algorithm that aims to achieve low power under throughput constraints. Unlike existing hardware-controlled power management algorithms, SmartBIPS uses real run-time data based on the operating power and performance history of the task set on the chip, and dynamically selects the operating power modes of the different cores. The impact of the chip level shared resources (like shared memory bottlenecks) on computation times and throughputs is captured in history tables for the different cores, and these data are used by SmartBIPS to predict the throughputs of the cores under new DVFS levels that the algorithm may assign to the cores for power optimal operation. Cores which execute memory bound tasks, or are otherwise in *stall* modes for considerable times, are dynamically slowed down to save power without impacting the throughput, whereas cores with high computation throughputs are operated at high voltage (and hence, frequency) levels. This ensures that cores which have high throughput operate at high power and performance points, and power reduction is mostly carried out for low performance tasks on other cores. In order to study the relative merits of our algorithm with respect to similar existing ones, we have implemented the chip-wide DVFS and MaxBIPS [68] algorithms as well. Our experimental setup includes a SPARCV9 based cycle-accurate chip multi-threading multi-core simulation platform, CASPER [69], and a suite of Network Packet Processing benchmarks called Embedded Network Processing (ENePBench) that we have developed (discussed in Chapter 3). Our results show that on average, SmartBIPS achieves a 41.3% improvement in power savings compared to

MaxBIPs, and a 19.8% improvement in throughput per unit power. This analysis is obtained by running network packet processing benchmarks on CASPER. The throughput improvement of SmartBIPS with respect to chip-wide DVFS is 16.7% at a cost of 1.2 times higher power dissipation. MaxBIPS achieves a 61% throughput improvement at a cost of 2.1 times higher power with respect to chip-wide DVFS.

5.6 Power Management Unit Architecture

For multi-core processors, the global power manager monitors activities in all the cores and take proper voltage-frequency mode-setting decisions with the target of enforcing a chip-level performance budget at the minimal power dissipation point. Figure 5-3 shows our proposed hierarchical power management architecture at the local intra-core and global inter-core levels. Any component or an entire core that can either be clock gated or power gated or voltage-frequency scaled, is a power saving candidate (PSC). Above the dashed line, the local power management unit (LPMU) operates inside a core, observes the content of the power status registers (PSRs) which are associated with different PSCs, executes a power saving algorithm based only on clock-gating and power-gating, and modifies the value in the corresponding power control registers (PCRs) to activate or deactivate power saving.

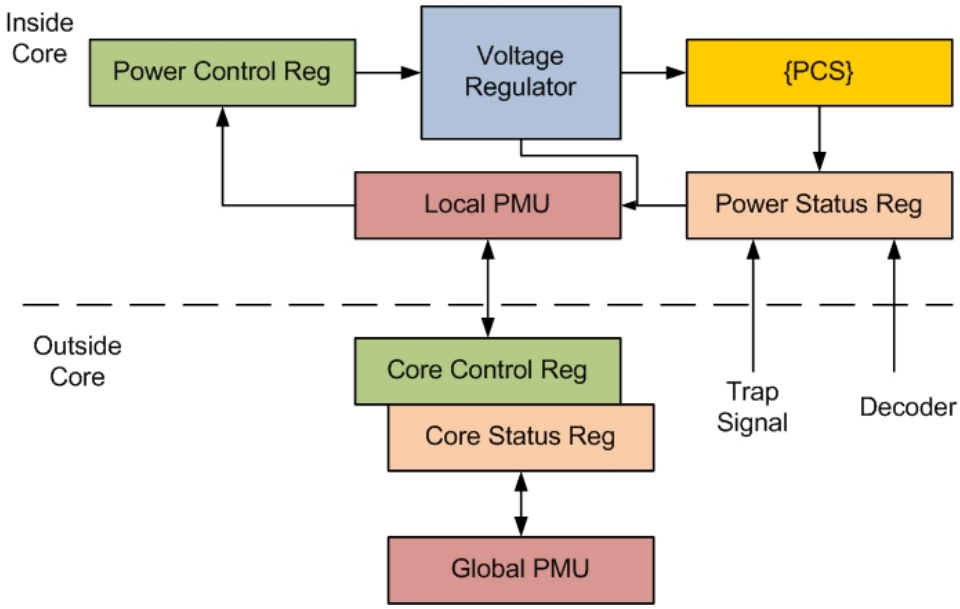

Figure 5-3: Architectural overview of autonomous hardware power saving scheme

The PSRs inside the cores are updated by the trap logic and the decoder, which signal the impending activation of the power saving candidate when certain interrupts have to be serviced or certain instructions are decoded. Similarly, the power saving candidates themselves can update their PSRs to signal the impending power saving due to prolonged inactivity (idle or blocked status) which is better observed locally inside a core. The LPMU algorithm that we have implemented is based on delay monitoring; specific PSCs have specific delay thresholds for clock-gating and power-gating, and after the PSCs have been idle for longer than these thresholds, power saving is either activated by clock-gating or power-gating. The clock-gating threshold is set to 1 clock cycle, while power-gating thresholds are longer and specific to the PSC. The PCR contents are read by the on-chip analog voltage and clock regulators which use that data to implement power-gating and clock-gating on the power saving candidates.

Below the dashed line and outside the cores, is the chip level GPMU which makes intelligent DVFS based power management decisions about the cores. The GPMU interacts with the cores through core status registers (CSRs) and core control registers (CCRs). We have used the above LPMU scheme for all three global power management algorithms discussed in this dissertation. Figure 5-4 shows details of the GPMUs interactions (CR and SR denote control and status registers respectively).

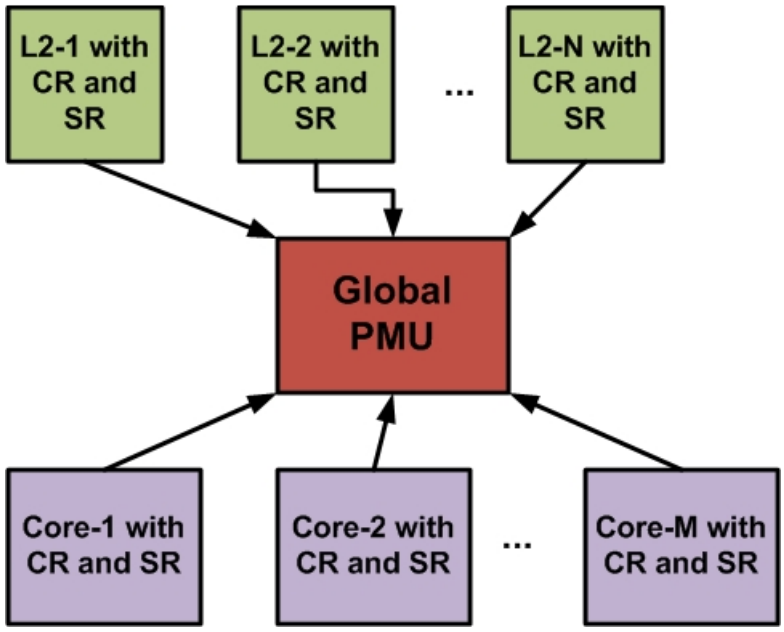

Figure 5-4: Interactions of Global Power Management Unit

5.7 The Experimental Setup

In order to evaluate the efficacies of SmartBIPS, MaxBIPS and chip-wide DVFS algorithms, we use CASPER [69]. For these experiments, we have modeled the architectural parameters to include 4 cores with a single hardware thread per core (virtual processor), a register file size of 160 registers, instruction translation lookaside buffer (TLB) size of 128, cache-size and coherence

protocols, L1 data cache of size 8KB and instruction caches of size 16KB, and instruction queue size of 1. The shared memory subsystem is configured as a shared L2 cache of size 1GB with 4 banks. The interconnection network is a crossbar. The processor architecture is homogeneous many-core architecture. The micro-architecture of each core is same as described in Chapter 4, containing IFU, BRU, EXU and LSU, L1 I/D$, LMQ, SB and I/D TLB and so on.

We use the ENePBench application suite used as the benchmark application in our experiment.

5.8 Existing Global Power Management Policies

Two existing hardware-controlled global power management policies – chip-wide DVFS and MaxBIPS are implemented and the results are compared with those of our novel SmartBIPS algorithm. Note that all these algorithms continuously re-evaluate the voltage-frequency operating levels of the different cores, once every evaluation cycle. If not explicitly stated, one evaluation cycle corresponds to 1024 processor clock cycles in our simulations.

The DVFS based GPMU algorithms rely on the assumption that when a given core switches from power mode A (voltage_A, frequency_A) in time interval N to power mode B (voltage_B, frequency_B) in time interval N+1, the power and throughput in time interval N+1 can be predicted using Equation 5-3 shown in Table 5-1. Note that the system frequency needs to scale along with the voltage to ensure that the operating frequency meets the timing constraints of the circuit whose delay changes linearly with the operating voltage [76]. This assumes that the workload characteristics do not change from one time interval to next one, and

there are no shared resource dependencies between tasks and cores. Table 5-1 explains the dependencies of power and throughput on the voltage and frequency levels of the cores.

Table 5-1: Relationship of power and throughput in time interval N and N+1

Time Interval	**N**	**N+1**
Mode	(v, f)	(v', f') f' = f (v'/v)
Throughput	T	T' = T*(f'/f) (5-2)
Dynamic Power	P	$P' = P*(v'/v)^2*(f'/f)$ (5-3)

Our power modes are defined as follows: VF_mode1 (1.2V, 2GHz), VF_mode2 (1.0V, 1GHz), and VF_mode3 (0.8V, 0.5GHz). These voltage-frequency pairs have been verified using the experimental setup. Note that performance predictions of the existing GPMU algorithms to be discussed in this section do not consider the bottlenecks caused by shared memory access between cores.

5.8.1 Chip-wide DVFS

Chip-wide DVFS is a global power management scheme that monitors the entire chip power consumption and performance, and enforces a uniform voltage-frequency operating point for all cores to minimize power dissipation under a chip-wide throughput budget. This approach does not need any individual information about the power and performance of each core, and simply relies on entire chip throughput measurements to make power mode switching

decisions. As a result, one high performance core can push the entire chip over throughput budget, thereby triggering DVFS to occur across all cores on chip. A scaling down of voltage and frequency in cores which are not exceeding their throughput budgets will further reduce their throughputs. This may be undesirable, especially if these cores are running threads from different applications which run at different performance levels.

Table 5-2: Pseudo Code of Chip-wide DVFS (this algorithm continuously executes once every evaluation cycle)

A. Get_current_core_dvfs_level;

B. For all $Cores_i$ {

    a. Get power dissipated by $Core_i$ in the last clock cycle;

    b. Get effective throughput of $Core_i$ in the last clock cycle;

    c. Sum up cumulative power dissipated by all cores in the last clock cycle;

    d. Sum up cumulative throughput of all cores in the last clock cycle;

}

C. If (Overall effective throughput of all cores > throughput budget) {

    a. if (current_core_dvfs_level > lowest_dvfs_level) {

        i. Lower down current_core_dvfs_level to next level;

    }

}

D. For all $Cores_i$ {

    a. Update every core's new dvfs level;

}

5.8.2 MaxBIPS

The MaxBIPS algorithm [68] monitors the power consumption and performance at the global level and collects information about the entire chip throughput, as well as the throughput contributions of individual cores. The power mode for each core is then selected so as to minimize the power dissipation of the entire chip, while maximizing the system performance subject to the given throughput budget. The algorithm evaluates all the possible combinations of power modes for each core, and then chooses the one that minimizes the overall power dissipation and maximizes the overall system performance while meeting the throughput budget by examining all voltage/frequency pairs for each core. The cores are permitted to operate at different voltages and frequencies in MaxBIPS algorithm. A linear scaling of frequency with voltage is assumed in MaxBIPS [68].

Based on Table 5-1, the MaxBIPS algorithm predicts the estimated power and throughput for all possible combinations of cores and voltage/frequency modes (vf_mode) or scaling factors and selects the (core_i, vf_mode_j) that minimizes power dissipation, but maximizes throughput while meeting the required throughput budget.

Table 5-3: Pseudo code of MaxBIPS (this algorithm continuously executes once every evaluation cycle)

A. Define_power_mode_combinations;

B. Initialize Min_power;

C. Initialize Max_throughput;

D. Initialize Selected_combination;

--voltage frequency (power mode) combinations for different cores

E. For all $Cores_i$ {

a. dvfsLevel = Get current DVFS level of $Core_i$;

b. Get power dissipated by $Core_i$ in the last clock cycle;

c. Get effective throughput of $Core_i$ in the last clock cycle;

}

F. For all $Power\_Mode\_Combinations_j$ {

A. For all $Cores_k$ {

a. Calculate predicted throughput value of core k in combination_j;

--Using power_mode_combination, Equation (5-2)

b. Calculate predicted power value of core k in combination_j;

--Using power_mode_combinations, Equation (5-2)

c. Accumulate predicted throughputs of all cores in combination_j;

d. Accumulate predicted power dissipations of all cores in combination_j;

}

B. If (overall_predicted_throughput of all cores <= throughput budget) {

a. If (Max_throughput < overall_predicted_throughput of all cores) {

i. Max_throughput = overall_predicted_throughput of all cores;

ii. Min_power = overall_predicted_power of all cores;

iii. Selected_combination = j;

}

b. If (Max_throughput == overall_predicted_throughput of all cores) {

iv. Max_throughput = overall_predicted_throughput of all cores;

v. If (Min_power >= overall_predicted_power of all cores)

vi. Min_power = overall_predicted_power of all cores;

vii. Selected_combination = j;

}

}

}

E. For all $Cores_i$ {

Update every core's new dvfs level with values in Selected_combination;

}

5.8.3 SmartBIPS Power Management Scheme

Most event-driven systems are non-deterministic, and hence power management decisions have to be made based on predictions of future workloads. A promising concept in power management predictive techniques for processors is to explore the past history of performance in order to make reliable predictions about future behavior.

In our proposed SmartBIPS method, the global power management unit periodically monitors the power and throughput of each core in every time interval

(a pre-set number of clock cycles, typically 1024) and predicts the optimal operating modes of the cores for the next time interval based on recent history of the system behavior and performance. SmartBIPS captures real run-time data based on the operating power and performance history of the task set on different cores on the chip in history tables, and uses it to make dynamic decisions about selecting operating power modes of the cores in the near future. There exists separate power and throughput entries in these history tables for every core and for every power (voltage-frequency) mode the core operated in. The user pre-defines a certain number of time intervals over which the performance and power numbers at different DVFS levels are averaged and stored in the history tables. The impact of the chip level shared resources (like shared memory bottlenecks) on computation times, throughputs and power dissipation on the different cores is automatically captured and encoded in the history tables. Hence, cores which execute memory bound tasks, or are otherwise in stall modes for considerable times, are dynamically slowed down to save power without impacting the throughput, whereas cores with high computation throughputs are operated at high voltage (and hence, frequency) levels. This ensures that cores which have high throughput operate at high power and performance points, and power reduction is mostly carried out for low performance tasks on other cores.

Different depths of history tables and different history data sampling methods are also implemented in order to observe the sensitivity of the results with different parameters. For 4096 clock cycles in one time interval (history table depth of 4096), results show that there is very little difference for power and

throughput of SmartBIPS with respect to 1024 clock cycles in every time interval. Moreover, sampling the parameters of interest every 128 clock cycles, and random sampling, within one time interval have been implemented as well. Results show that sampling every 128 clock cycles does not improve power saving and throughput per unit power of SmartBIPS compared to MaxBIPS; random sampling improves power saving by about 10% on average, but no improvement in throughput per unit power of SmartBIPS is achieved with respect to MaxBIPS. Sampling every clock cycle improves both power saving and throughput per unit power compared to chip-wide DVFS and MaxBIPS.

A scaling factor α (with values between 1.0 and 1.5) is empirically defined to control throughput reduction in SmartBIPS. Only if the history table throughput data is α times greater than MaxBIPS predicted throughput at a certain power level, a scaling of lower voltage/frequency level is allowed in SmartBIPS; this achieves a high throughput per unit power while saving power. Table 4 shows the average power, average throughput, and throughput per unit power of SmartBIPS with different values of α at different power levels. Note that throughput is typically measured in terms of number of instructions per cycle (IPC). However, because of DVFS the clock period for a core can potentially change in every evaluation cycle. Hence, we use the metric of instructions per nanosecond (IPnS) to capture throughput. Also, α = 1 for levels are not explicitly mentioned in Table 5-4.

Table 5-4: Average power, average throughput, T/P with different α values at different power levels in SmartBIPS

10000 clk cycles, time interval is 1024, 90% T_budget	**Average power in one time interval (W)**	**Average throughput in one time interval (IPnS)**	**Throughput per unit power**
With α 1.25 at VF_mode3	0.205	0.276	1.345
With α 1.2 at VF_mode3	0.205	0.276	1.345
With α 1.1 at VF_mode3	0.205	0.276	1.345
With α 1.25 at VF_mode2 and VF_mode3	0.208	0.285	1.370
With α 1.25 at VF_mode2 and with α 1 at VF_mode3	0.123	0.21	1.68
MaxBIPS	0.209	0.295	1.411

From Table 5-1 we notice that when α is 1 at level 3 and α is 1.25 at level 2, SmartBIPS can save power, keep a relative high throughput and give a high throughput per unit power with respect to those of MaxBIPS.

Table 5-5: Pseudo-code of SmartBIPS (this algorithm continuously executes once every evaluation cycle)

A. Define_power_mode_combinations;--like Equation (5-3)

B. Initialize Min_power;

C. Initialize Max_throughput;

D. Initialize Selected_combination;

--voltage frequency (power mode) combinations for different cores

E. For all $Cores_i$ {

a. dvfsLevel = Get current DVFS level of $Core_i$;

b. Get power dissipated by $Core_i$ in the last time interval;

c. Get effective throughput of $Core_i$ in the last time interval;

}

F. For all Power_Mode_Combinationsj {

a. For all $Cores_k$ {

A. Calculate predicted throughput of core i in combination_j;

--Using power_mode_combinations, Equation (5-3)

B. Calculate predicted power dissipation of core i in combination_j;

--Using power_mode_combinations, Equation (5-3)

C. Accumulate predicted throughput values of all cores;

D. Accumulate predicted power values of all cores;

}

b. If (overall_predicted_throughput of all cores <= throughput budget) {

A. If (Max_throughput < overall_predicted_throughput of all cores) {

i. Max_throughput = overall_predicted_throughput of all cores;

ii. Min_power = overall_predicted_power of all cores;

iii. Selected_combination = j;

}

B. If (Max_throughput = overall_predicted_throughput of all cores) {

i. Max_throughput = overall_predicted_throughput of all cores;

ii. If (Min_power >= overall_predicted_power of all cores) {

a. Min_power = overall_predicted_power of all cores;

b. Selected_combination = j;

}

}

} -- upto here we are following MaxBIPS

--start of code unique to SmartBIPS

c. For all $Cores_i$ {

A. For all dvfs_levelj {

B. Check History Table entries for throughput at dvfs_level j;

C. Get average_history_throughput of core i at dvfs_level j;

}

}

d. For all $Cores_i$ {

A. Initialize predicted dvfs level of MaxBIPS to core_i;

B. If (core_i_next_dvfs_level == lowest_dvfs_level) {

i. Go to next core;

}

C. Else if (average_history_throughput of core i at predicted dvfs_level > predicted throughput value of core i by MaxBIPS *factor_alpha) { -- factor_alpha is an empirical paramter between 1.0 and 1.5

i. Lower down current predicted dvfs_level of core i to next level;

ii. Get average_history_power of core i at predicted dvfs_level;

}

D. Else keep current predicted dvfs_level of core i;

}

e. For all $Cores_i$ {

A. Update every core's new dvfs level;

}

}

In the SmartBIPS algorithm, the actual power dissipation and throughput for the chosen DVFS level combination for the different cores (as stored in the history table) are found using the processor simulator CASPER (discussed in Chapter 4), which executes instructions between every pair of consecutive time interval boundaries when the global power manager re-evaluates the DVFS levels of the different cores.

5.9 Experimental Results

In this section we show throughput and power comparisons of the three power management algorithms, followed by a similar comparison with a modification of the chip-wide DVFS algorithm – the chip-wide DVFS throughput is lower-bounded to of 60% of its peak throughput. We conclude that lower-bounding the throughput of chip-wide DVFS effectively lowers its throughput per unit power metric below those obtained by SmartBIPS and MaxBIPS. Finally we compare the average power, average throughput, and average throughput per unit power, average energy and average latency of three discussed policies.

In Figure 5-5 and Figure 5-6, we show the power and throughput data respectively (with a throughput budget constrained to at 90% of peak throughput with any voltage and frequency scaling) for our three discussed policies for packet type 3 (TYPE3) which is a typical representative of all other packet types. Values on the X-axis correspond to the number of evaluation cycles, where one evaluation cycle is the time period between consecutive runs of the power management algorithms. Where not explicitly stated, one evaluation cycle corresponds to 1024 processor clock cycles in our simulations. In Figure 5-5, the X-axis represents number of clock cycles and the Y-axis represents power (W). In Figure 5-6, the Y-axis represents throughput (in instructions per nanosecond - IPnS).

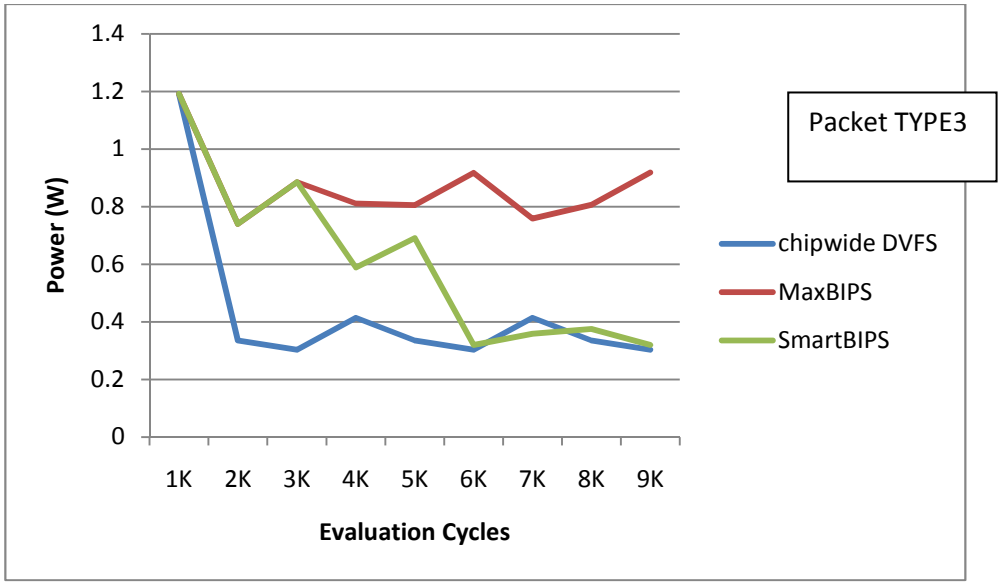

Figure 5-5: Power dissipation data for three global power management policies for packet type TYPE3 with throughput budget constrained to 90%

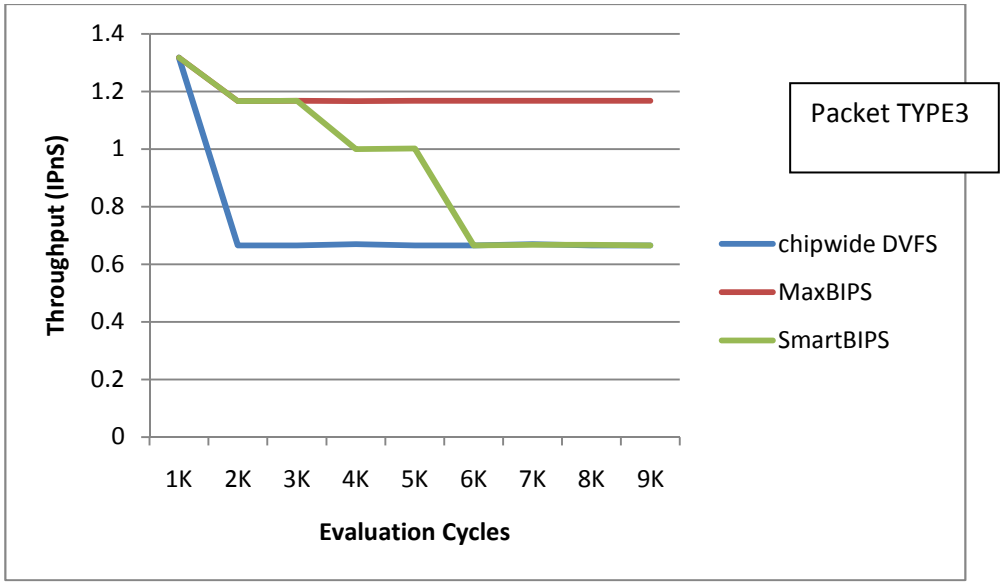

Figure 5-6: Throughput data for three global power management policies for packet type TYPE3 with throughput budget constrained to 90%

As Figure 5-5 shows, the power consumption of MaxBIPS is much higher than those of SmartBIPS and chip-wide DVFS (the latter being the lowest in power dissipation among the three methods). However the throughput of

MaxBIPS is also higher than those of the other two policies. SmartBIPS has lower throughput than MaxBIPS but higher than that of chip-wide DVFS.

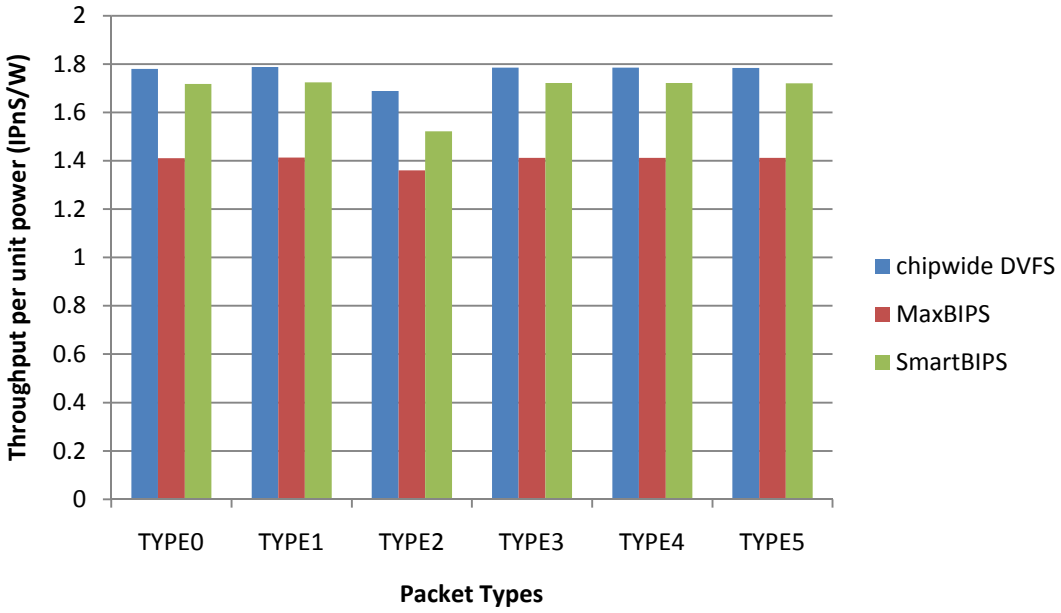

Figure 5-7: Throughput per unit power for all packet types for the 3 methods

Figure 5-7 depicts the throughput per unit power (T/P) data for the three methods. While chip-wide DVFS has the highest T/P values for the different packet types, the SmartBIPS T/P is greater than that of MaxBIPS and is very close to the T/P of chip-wide DVFS. Note that high T/P value for chip-wide DVFS arises from the fact that power dissipation in this scheme is substantially lower than other schemes, and not because the throughput is high. When implementing power management by chip-wide DVFS, any increase in the throughput of a single core over a target threshold triggers chip-wide operating voltage (and hence, frequency) reductions in all cores, to save power. Hence, once the overall throughput exceeds the budget, all the cores have to adjust their power modes to

a lower level. While this method reduces the chip-wide power dissipation substantially, it also leads to excessive performance reductions in all cores as shown in Figure 5-6.

A modification of the chip-wide DVFS algorithm required for achieving high performance is to assign a lower bound of throughput. Figure 5-8 and Figure 5-9 show the power and throughput data respectively (with a lower bound of throughput budget constrained to at 60% of peak throughput with all voltage-frequency levels) for chip-wide DVFS for packet type 3 (TYPE3). The power consumption and throughput of chip-wide DVFS are higher than those of MaxBIPS and SmartBIPS due to the lower bound of throughput which does not allow chip-wide DVFS to scale all the cores to lower voltage-frequency levels in order to guarantee the system performance. However the throughput per unit power of chip-wide DVFS is lower than those of the other two policies as Figure 5-10 demonstrates. SmartBIPS has the highest throughput per unit power compared to MaxBIPS and chip-wide DVFS. Table 5-6 shows the power, throughput, and throughput per unit power of chip-wide DVFS with and without lower bound on throughput.

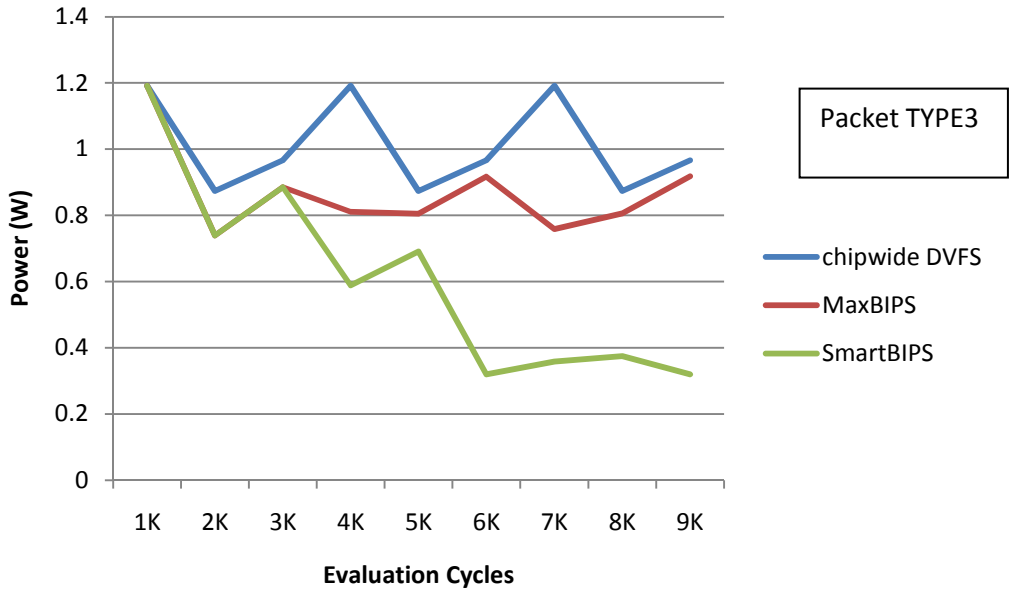

Figure 5-8: Power dissipation for three global power management policies (chip-wide DVFS throughput budget constrained to 60% of peak throughput)

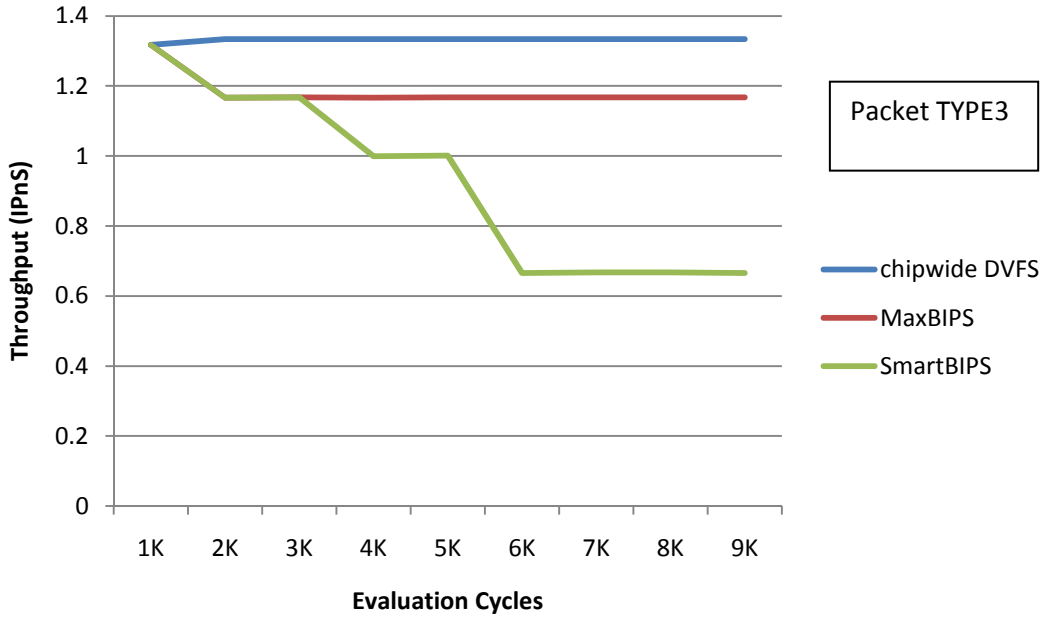

Figure 5-9: Throughput observed in three global power management policies (chip-wide DVFS throughput budget constrained to at 60% of peak throughput)

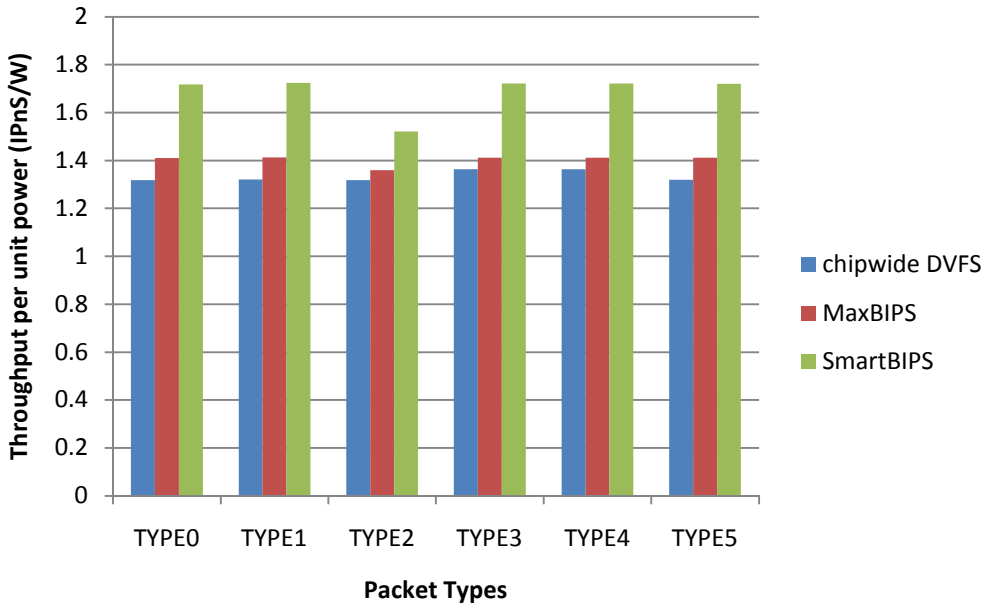

Figure 5-10: Throughput per unit power data (chip-wide DVFS with lower bound throughput)

Table 5-6: Power, throughput, throughput per unit power of chip-wide DVFS with and without lower bound on throughput

	With lower bound 60% of peak T			**Without lower bound of throughput**		
	Power in one time interval (W)	**Throughput in one time interval (IPnS)**	**Throughput per unit power (IPnS/W)**	**Power in one time interval (W)**	**Throughput in one time interval (IPnS)**	**Throughput per unit power (IPnS/W)**
chip-wide DVFS	0.252	0.332	1.318	0.105	0.184	1.752

In summary, experimental data show that when chip-wide DVFS is not enabled with lower bound of throughput, MaxBIPS has the highest throughput. However, SmartBIPS has a better throughput per unit power and saves more power than MaxBIPS. Although chip-wide DVFS gives the highest throughput per unit power, its throughput, on average, is lower than that of SmartBIPs, which can be a constraining factor in high throughput systems that require throughputs close to the budget. When chip-wide DVFS is lower-bounded to 60% of peak throughput achievable by chip-wide DVFS, it produces the highest throughput and consumes the highest power among all the three methods. This yields the lowest throughput per unit power for chip-wide DVFS, and SmartBIPS saves more power and achieves the highest throughput per unit power compared to the other two policies. Table 5-7 shows the relevant experimental results of three policies with different packet types.

Table 5-7: Power, throughput, throughput per unit power of three policies for different packet types

	chip-wide DVFS without lower bound			**MaxBIPS**			**SmartBIPS**		
	P (W)	**T (IPnS)**	**T/P (IPnS/W)**	**P (W)**	**T (IPnS)**	**T/P (IPnS/W)**	**P (W)**	**T (IPnS)**	**T/P (IPnS/W)**
TYPE0	3.72	6.61	1.78	7.53	10.62	1.41	4.24	7.28	1.72
TYPE1	3.72	6.65	1.79	7.54	10.65	1.41	4.24	7.31	1.72
TYPE2	3.93	6.65	1.69	7.83	10.65	1.36	5.47	8.32	1.52

TYPE3	3.72	6.64	1.78	7.54	10.64	1.41	4.24	7.30	1.72
TYPE4	3.72	6.64	1.78	7.54	10.63	1.41	4.24	7.30	1.72
TYPE5	3.72	6.63	1.78	7.53	10.64	1.41	4.24	7.29	1.72
Average	**3.75**	**6.64**	**1.76**	**7.58**	**10.64**	**1.40**	**4.45**	**7.47**	**1.68**

Table 5-8 compares the power savings and T/P gains of SmartBIPS compared to MaxBIPS. Results show that on average, SmartBIPS achieves a 41.3% improvement in power savings compared to MaxBIPS, and a 19.8% improvement in throughput per unit power with respect to MaxBIPS.

Table 5-8: Power saving and throughput per unit power improvement of SmartBIPS with respect to MaxBIPS

Packet Types	**Power saving**	**Throughput per unit power**
TYPE 0	43.72%	21.8%
TYPE 1	43.72%	22.0%
TYPE 2	30.2%	11.8%
TYPE 3	43.71%	21.9%
TYPE 4	43.72%	21.9%
TYPE 5	43.72%	21.9%
Average	**41.3%**	**19.8%**

Table 5-9 shows the average power, average throughput, average throughput per unit power, average energy and average latency (execution time)

of three discussed policies while running about 7300 instructions for all the packet types (averaging is done over all packet types). Results show that on average, chip-wide DVFS consumes 17.7% more energy than MaxBIPS and has 2.34 times its latency. SmartBIPS consumes 8.2% more energy than MaxBIPS and takes 1.85 times longer execution time. However, SmartBIPS achieves a 41.3% improvement in power savings and a 19.8% improvement in throughput per unit power with respect to MaxBIPS. Hence SmartBIPS is an ideal candidate for use in applications with relatively high throughput requirements than what chip-wide DVFS can provide, and with cooling capacity limits lower than what MaxBIPS demands.

Table 5-9: Average power, average throughput, average throughput per unit power, average energy, and average execution time of three discussed policies

	P_average (W)	**T_average (IPnS)**	**T/P_average (IPnS/W)**	**Energy_average (nJ)**	**Average Latency (nS)**
chip-wide without lower bound	3.75	6.64	1.77	3.371	34816
MaxBIPS	7.58	10.64	1.40	2.864	14848
SmartBIPS	4.44	7.47	1.68	3.100	27477

5.10 Conclusion

This chapter of this dissertation presents SmartBIPS, a new algorithm for hardware controlled dynamic power management in embedded multi-core

processors executing real-time constrained high performance applications. SmartBIPS minimizes power dissipation while maximizing the chip level performance, subject to throughput constraints. The proposed SmartBIPS algorithm is based on chip-level monitoring, control and dynamic management of power for multiple cores. The global power management unit (GPMU) is aware of the activities of all the cores in a system, captures the throughput and power dissipation history of every core in shifting temporal windows, and makes intelligent prediction for power management based on recent workload power-performance history. Performance bottlenecks due to inter-core sharing of global resources, including on-chip interconnection networks and higher level cache memories, are captured in the history tables used by the GPMU.

Results show that on average, SmartBIPS achieves a 41.3% improvement in power savings and a 19.8% improvement in throughput per unit power with respect to MaxBIPS. This analysis is obtained using CASPER [69], a cycle-accurate simulation platform for multi-core processors, using a network packet processing benchmark that we have developed. The throughput improvement of SmartBIPS with respect to chip-wide DVFS is 16.7% at a cost of 1.2 times higher power dissipation. MaxBIPS achieves a 61% throughput improvement at a cost of 2.1 times higher power with respect to chip-wide DVFS.

These results encourage us to believe in the potential applicability of hardware controlled dynamic power management for embedded multi-core processors with global monitoring and control. To the best of our knowledge, the SmartBIPS algorithm presented in this dissertation is the first to consider shared

resource constraints for dynamic power management. In the future we will study other hardware-controlled power management strategies. In addition we plan to design the hardware circuits which implement the different hardware-controlled power management schemes, and evaluate the hardware area, power and performance trade-offs.

CHAPTER 6: MODELING OF THROUGHPUT AND POWER DISSIPATION OF CORES

6.1 Theory of Statistical Curve Fitting

The generalized linear regression models of n-variables are shown in the Equation 6-1 and Equation 6-2 respectively:

$$Y = c_0 + \sum_{i}^{n} c_i x_i + \varepsilon \qquad (6-1)$$

$$Y = c_0 + \sum_{i}^{n} c_i x_i + \sum_{i}^{n}\sum_{j}^{n} c_{i,j} x_i x_j + \sum_{i}^{n}\sum_{j}^{n}\sum_{k}^{n} c_{i,j,k} x_i x_j x_k + \ldots + c_{1,2,\ldots,n} x_1 x_2 \ldots x_n + \varepsilon \qquad (6-2)$$

In our case, the different dependent variable (denoted by Y) is CPI per thread, CPI per core or total power dissipation of cores. The micro-architectural parameters listed in Table 6-1 constitute the predictor variables $x_1, x_2, \ldots, x_n$. Our objective is to perform regression using our training dataset and then derive the correlation coefficients $c_0, c_1, \ldots, c_n$ in Equation 6-1 and Equation 6-2 such that we can achieve <10% error of estimates.

6.2 Micro-architectural Parameters used in statistical curve-fitting

Table 6-1 shows the set of micro-architectural parameters that we tune to derive optimized designs of cores. Throughput and power dissipation of a core primarily depends on N_T number of threads, the L1 instruction and data cache

size, associativity and line sizes and the miss queues. Each thread has its own register file and hence contributes significantly to power dissipation of a core. Also, as N_T is scaled, throughput per thread decreases as each thread needs to wait as many cycles before its next instruction is issued to the D-stage. However, throughput per core might increase. Our processor model follows a *write-through* scheme for the store instructions. Moreover, stores are serialized following the Total Store Order (TSO) model explained in [56]. The TSO model is implemented through the store buffer which serializes all the stores of a hardware thread. Off-core L2 traffic hence consists of stores, instruction and data cache misses. Instruction misses are detected in I$ in the F-stage of the pipeline and then enqueued into the MIL. Duplicate instruction misses are blocked in the core and never forwarded to L2. Data misses are detected in D$ in the M-stage in load store unit and are enqueued in LMQ. Duplicate data misses are also blocked in the core. Since outgoing packets from all the cores are first enqueued in the L2 queues and then arbitrated into the processing controller of the L2 cache, the total L2 cache access time depends on the overall L2 traffic. In the meantime, the core has to wait and keep asserting the interconnect signals to check whether the required L2 reply has arrived. Hence, the throughput and power dissipation of a core depends on L2 queue size and the L2 access time which are also included in the core optimization process. Address Space Identifier (ASI) register load, store and atomic instructions are all processed through the ASI queue. Hence, ASI queue is also an important micro-architectural feature affecting throughput and power dissipation of a core.

Table 6-1: Micro-architectural Parameters of a Multi-threading Core

Name	Range	Increment	Description
1. N_T	1 to 16	Power of 2	Threads per core
2. Load Miss Queue (LMQ) Size Per Thread	1 to 16	Power of 2	Used to enqueue all the D$ misses
3. SB Size Per Thread	1 to 16	Power of 2	Used to serialize the store instructions following the TSO model [56]
4. L1 ICache Associativity	2 to 8	Power of 2	Set-associativity of I$
5. L1 ICache Line Size	8 to 64	Power of 2	Block size of I$
6. L1 ICache Size	1KB to 64KB	Power of 2	Total I$ size
7. L1 DCache Associativity	2 to 8	Power of 2	Set-associativity of D$
8. L1 DCache Line Size	8 to 64	Power of 2	Block size of D$
9. L1 DCache Size	1KB to 64KB	Power of 2	Total I$ size
10. MIL Size Per Thread	1 to 16	Power of 2	Used to enqueue the I$ misses

11. ASI Queue Size Per Thread	1 to 16	Power of 2	Used to serialize all Address Space Identifier register reads/writes
12. L2 Access Time	25 to 1000	Incremented by 1	Hit latency of L2 cache
13. L2 Input Queue Size per Core	4 to 16	Power of 2	Used to enqueue all the core-to-L2 ifill/load/store packets

6.3 Regression Models and Error Analysis

Although there is no direct way of knowing the best length of training dataset, the rule of thumb in case of both linear and non-linear multiple regressions is to get at least 10 times as many training cases as input variables. These way inherent problems such as over-fitting or under-fitting can be avoided in multiple non-linear regressions. However, with noise free targets, twice as many training cases as input variables would be more than adequate. In our case for each packet type, we have collected 100 sets of data which contains CPI per thread, CPI per core and power dissipation of the cores.

Note that CPI per thread is modeled to accurately predict the processing time of a packet which is mapped to a hardware thread. This packet processing time is used during design space exploration to evaluate whether all packets in the system are meeting the real time constraints. After careful consideration of

the CPI and power dissipation for each packet types, we found that CPI per thread and power dissipation of packet types TYPE0, TYPE1 and TYPE2 are linearly related to the micro-architectural parameters of the multi-threaded core as shown by Equation 6-3.

$$Y = c_0 + c_1 * Threads + c_2 * LMQ + c_5 * DC_{Size} + c_6 * DC_{linesize} + c_7 * ASIQ + c_8 * IC_{Size} + c_9 * IC_{Linesize} + c_{10} * L2_{AccessTime} + c_{11} * L2_{Q_{Size}} \quad (6-3)$$

The small number of store instructions (<1%) in these compute bound packets do not have a major impact on performance or power dissipation and hence store buffer size does not appear in the statistical model. Similarly, due to low number of memory accesses in these packet types, higher order non-linear terms comprising of L2 cache access time and others do not appear in the model either.

The values of the correlation coefficients, corresponding variables and model parameters are shown in Table 6-2, Table 6-3 and Table 6-4 for packet types TYPE0, TYPE1 and TYPE2 respectively. Note that R is the multiple correlations co-efficient which are the linear correlation between the observed and model predicted values of dependent variable. Large value indicates strong relationship. R^2 is the coefficient of determination which tells the percentage of time the variation is explained by the model.

Table 6-2: Linear Regression Correlation Coefficients and Model Parameters for TYPE0

Variable	Correlation	Power	CPI-	CPI-per-

	Coefficients	(mW)	per-strand	core
Constant	c_0	1.437	-9.176	5.744
Threads	c_1	0.086	2.658	-0.277
LMQ	c_2	-0.075	0.298	0.244
DC_Size	c_5	-0.011	0.123	-0.11
DC_Linesize	c_6	0.019	-0.069	-0.055
ASIQ	c_7	-0.092	-0.275	-0.104
IC_Size	c_8	0.012	0.084	-0.009
IC_Linesize	c_9	0.001	-0.012	0.000
L2_Access	c_{10}	0.003	0.438	0.074
L2_Q	c_{11}	-1.57	0.432	-0.028
Model Summary				
	Parameters			
	R	0.816	0.914	0.805
	R^2	0.665	0.836	0.649
	Std. Err. Of Estimates	0.354	7.141	1.537

Table 6-3: Linear Regression Correlation Coefficients and Model Parameters for TYPE1

Variable	Correlation	Power	CPI-per-	CPI-per-

	Coefficients	(mW)	strand	core
Constant	c_0	0.526	-10.014	6.015
Threads	c_1	0.021	3.656	-0.507
LMQ	c_2	-0.032	0.404	0.714
DC_Size	c_5	-0.006	0.070	0.010
DC_Linesize	c_6	0.004	-0.013	-0.109
ASIQ	c_7	-0.057	-3.826	0.372
IC_Size	c_8	0.004	1.234	-0.058
IC_Linesize	c_9	-0.002	0.162	-0.018
L2_Access	c_{10}	0.000	0.642	0.162
L2_Q	c_{11}	-0.035	-2.395	0.231
Model Summary				
	Parameters			
	R	0.796	0.907	0.839
	R^2	0.633	0.822	0.704
	Std. Err. Of Estimates	0.097	10.407	2.678

Table 6-4: Linear Regression Correlation Coefficients and Model Parameters for TYPE2

Variable	Correlation	Power	CPI-per-	CPI-per-

	Coefficients	**(mW)**	**strand**	**core**
Constant	c_0	0.270	-4.327	7.351
Threads	c_1	0.016	1.684	-0.215
LMQ	c_2	-0.013	0.292	0.063
DC_Size	c_5	-0.002	0.069	-0.032
DC_Linesize	c_6	0.003	-0.027	-0.012
ASIQ	c_7	-0.005	0.480	-0.751
IC_Size	c_8	8.61e-5	-0.046	0.086
IC_Linesize	c_9	4.053e-5	-0.030	-0.007
L2_Access	c_{10}	0.001	0.194	0.018
L2_Q	c_{11}	-0.026	0.445	-0.440
Model Summary				
	Parameters			
	R	0.792	0.964	0.751
	R^2	0.628	0.929	0.564
	Std. Err. Of Estimates	0.075	2.916	1.504

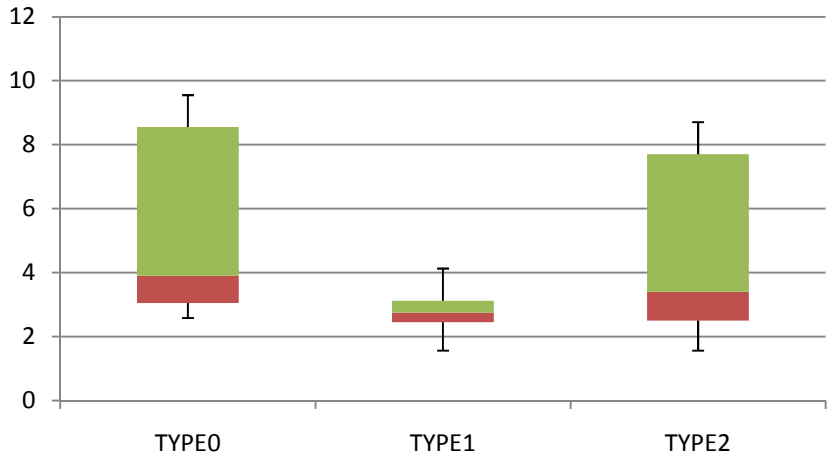

Figure 6-1: Error distribution of CPI-per-thread of packet types TYPE0, TYPE1 and TYPE2

The model in Equation 6-3 is validated using an error set comprising of 15 randomly chosen micro-architectural configurations for each of the packet types TYPE0, TYPE1 and TYPE2. The configurations in the error set were simulated in CASPER. The error distribution of CPI-per-thread as shown in Figure 6-1 is calculated by comparing the measured CPI-per-thread (CASPER) against predicted CPI-per-thread given by Equation 6-3. As the figure suggests, the error of the model was within the required limit of 10%. Also, the standard confidence of interval for each of the coefficients c_0 to c_{11} was measured and was observed to never cross zero value which suggests that all the coefficients were significant. Similarly, the error distribution of the power dissipation model of the cores for packet types TYPE0, TYPE1 and TYPE2 is shown in Figure 6-2. In this case also we found that the error of our model was less than 10%.

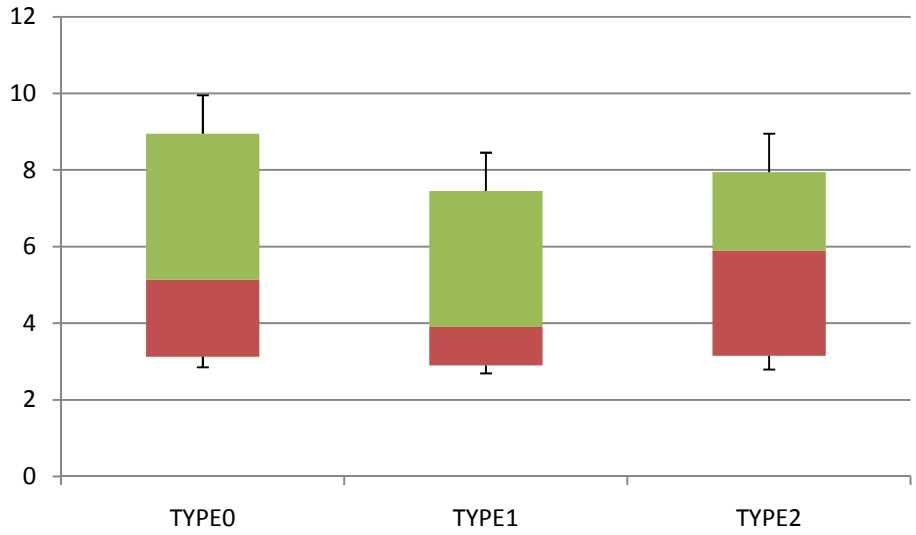

Figure 6-2: Error distribution of dynamic power dissipation (mW) of the cores processing packet types TYPE0, TYPE1 and TYPE2

Data bound packet types TYPE3 and TYPE4, on the other hand demonstrates a completely different behavior. In case of both TYPE3 and TYPE4 packet types, using only single factor terms in the linear regression model for either CPI per thread or power dissipation resulted in a high 15% standard error of estimate. Hence we included the 2-factor, 3-factor and 4-factor terms in our model which minimized the error of estimates and finally produced the prediction model equation given by Equation 6-4:

$$Y = b0 + b1 * Threads + b2 * LMQ + b3 * SB + b4 * MIL + b5 * dc_{size} + b6 * dc_{assoc} + b7 * dc_{linesize} + b8 * ASIQ + b9 * ic_{size} + b10 * ic_{assoc} + b11 * ic_{linesize} + b12 * L2_{access} + b13 * L2_{Q} + b14 * Threads * LMQ + b15 * Threads * SB + b16 * Threads * MIL + b17 * Threads * ASIQ + b18 * SB * L2_{access} * L2_{Q} + b19 * dc_{size} * dc_{assoc} * dc_{linesize} * LMQ * L2_{access} * L2_{Q} + b20 * ic_{size} * ic_{assoc} * ic_{linesize} * MIL * L2_{access} * L2\_Q \quad (6-4)$$

The correlation coefficients and model parameters of linear regression containing non-linear monomial are described in Table 6-5.

Table 6-5: Non-linear Regression Correlation Coefficients

Correlation Coefficients	**TYPE8**			**TYPE9**		
	Power (mW)	**CPI-per-strand**	**CPI-per-core**	**Power (mW)**	**CPI-per-strand**	**CPI-per-core**
b0	-5.13	-2.48	6.59	8.64	8.31	-8.62
b1	2.13	2.63	-1.59	-0.204e-6	1.328e-6	-0.442e-6
b2	-0.062e-6	-4.225e-6	0.481e-6	-0.044e-6	-3.316e-6	0.448e-6
b3	1.71	8.288e-1	-2.19	-2.87	-2.8	2.87
b4	-5.12	-2.8	6.59	8.64	8.3	-8.62
b5	-0.015e-6	-0.492e-6	0.069e-6	-0.013e-6	-0.665e-6	0.062e-6
b6	-1.28	-6.21e-1	1.65	2.16	2.1	-2.15
b7	0.028e-6	0.694e-6	-0.108e-6	0.026e-6	0.311e-6	-0.120e-6

b8	0.309e-6	-2.816e-6	-0.552e-6	0.308e-6	-0.794e-6	-0.395e-6
b9	-0.008e-6	0.870e-6	-0.112e-6	-0.009e-6	1.002e-6	-0.103e-6
b10	-1.28	-6.22e-1	1.65	2.16	2.1	-2.15
b11	-0.011e-6	-0.049e-6	0.019e-6	-0.010e-6	-0.022e-6	-0.026e-6
b12	-8.7e-2	1.37e4	8.99e4	9.3e4	2.1e6	-5.5e-2
b13	3.42	1.66	-4.39	-5.76	-5.5	-5.75
b14	0.001e-6	0.386e-6	-0.011e-6	0.000e-6	0.018e-6	-0.013e-6
b15	0.034e-6	-0.025e-6	-0.105e-6	0.034e-6	0.271e-6	-0.089e-6
b16	-2.13	-2.63	1.59	0.301e-6	1.828e-6	0.056e-6
b17	-0.078e-6	-0.330e-6	0.235e-6	-0.077e-6	-0.438e-6	0.219e-6
b18	5.43e-3	-856.05e-6	-5.6e-2	-580.87e-6	1.3e-2	3.41e-3
b19	-3.86e-7	-1.37e-7	8.83e-7	-2.90e-7	2.1e-7	1.46e-7
b20	2.75e-6	7.99e-5	-4.58e-6	2.46e-6	5.8e-7	-5.43e-7

Figure 6-3 (a) and (b) shows the model error distribution for the CPI per thread and power dissipation in case of packet types TYPE3 and TYPE4. Similar to packet types TYPE0, TYPE1 and TYPE2, randomly chosen sets of 15 micro-architectural configurations were used to compare measure and predicted values of CPI per thread and power dissipation in case of packet types TYPE3 and TYPE4 respectively. As evident from the figure, the prediction models could achieve less than 12% error.

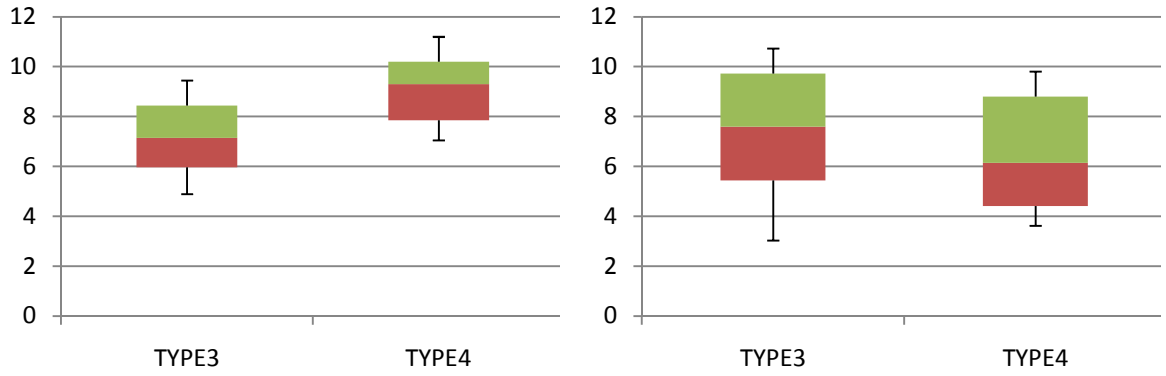

Figure 6-3: (a) Error distribution of CPI-per-thread model for cores of packet types TYPE3 and TYPE4 and (b) Error distribution of power dissipation model for cores of packet types TYPE3 and TYPE4

CHAPTER 7: EXPLORATION ALGORITHM

Various sources of routing and packet processing data show that in a day maximum number of incoming packets in a router is pass-through real-time packets. Hence we assume that the five types of IP packets, discussed in Table 3-2, arrive at the given distribution – among all the incoming packets per second, *60%* are TYPE0 packets, *25%* are TYPE1 packets, *5%* are TYPE2 packets, *5%* are TYPE3 packets and *5%* are TYPE4 packets. We believe that based on our observations this is a reasonable assumption. Moreover, our design flow can easily be tuned to consider other distributions of packet types. We also assume that the dynamic scheduler responsible for assigning the incoming packets to the respective customized different cores in the NeP is an ideal scheduler which is aware of the micro-architecture of the available cores in the system and is always able to satisfy schedulability of the system. Although scheduling can be a compute-intensive problem itself, exploring scheduling algorithms adds another complex dimension to our exploration problem and is out of the scope of this dissertation.

To efficiently explore the large and complex design space, we take the divide and rule approach. The structural characteristics of the micro-architecture enable us to divide the design space into the *core subsection* and the *memory subsection*. These two subsections are connected via the interconnection network which is a network of crossbar switches. Figure 7-1 shows the steps

involved in our exploration algorithm.

Step 1 (Regression Modeling): Given that packets can be scheduled to the cores in a NeP, we first attempt to explore core micro-architectures according to packet types as discussed in Table 3-2. To achieve the above, first we use CASPER to collect *training datasets* sampling the core micro-architectural design

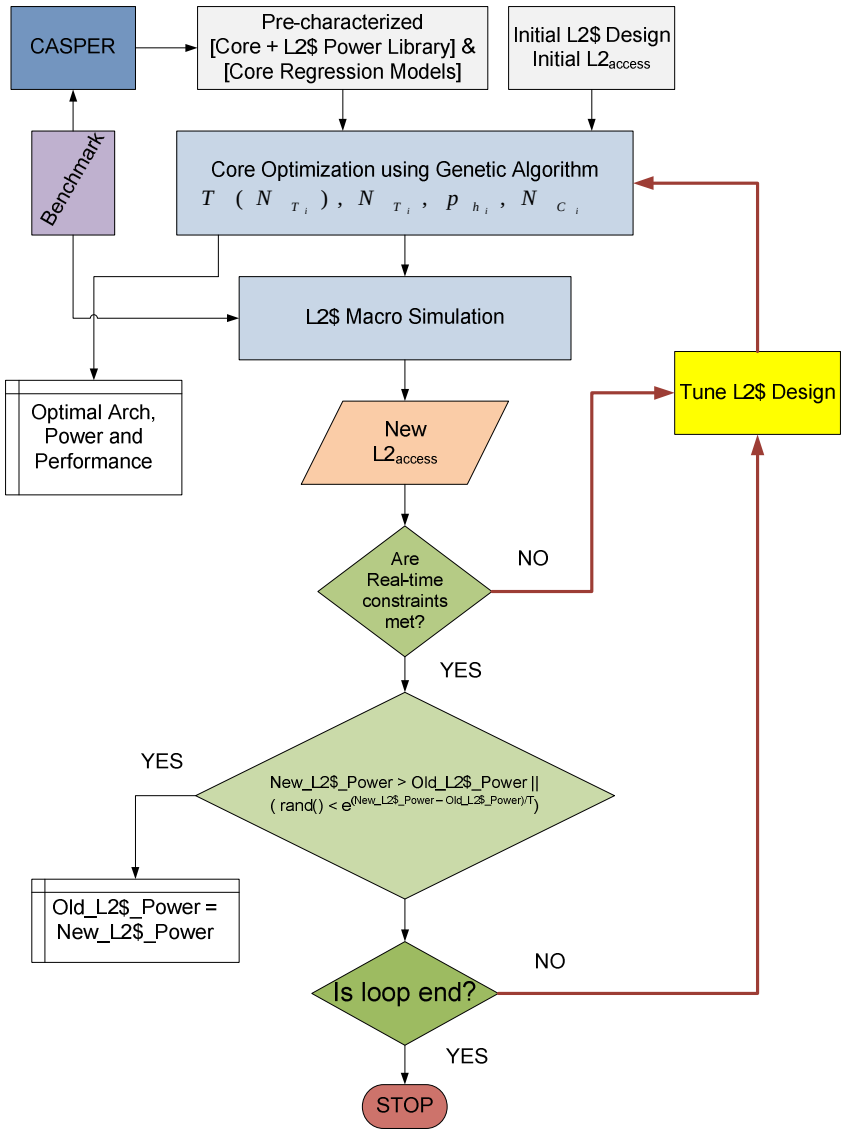

Figure 7-1: Exploration Algorithm

at uniform random intervals separately for each packet type. We then use

multiple non-linear regression [65] to derive statistical relations between the micro-architectural parameters described in Table 6-1 and cycles-per-instruction (CPI) of a thread, CPI of a core and total power dissipation of the core. Average power dissipation per clock cycle is calculated by dividing the total overall power dissipation of the core by the total number of simulated clock cycles. The steps involved, derived model parameters and model error analysis are described in Chapter 6.

Step 2 (Core Optimization): Derived statistical linear regressions of CPI per thread, CPI per core and power dissipation are used in a parallelized Genetic Algorithm based optimization engine called Fast Genetic Algorithm (FGA) [77] to generate a set of 10 best optimized core micro-architectures with minimal power dissipation. GA is a popular evolutionary meta-heuristic optimization algorithm used in a variety of optimization and search applications [78]. GA prototypes the characteristic processes of biological evolution, such as fitness, mutation and crossover.

In our design space, the micro-architectural parameters of a cores described in Table 6-1 are mapped to genes. A core which is expressed as a set of micro-architectural parameters represents a chromosome in the GA engine. We have used total 32 candidate designs in each of 400 generations in the GA engine. However, in majority of the cases the optimization algorithm converged within 180 generations. We also increased the default mutation rate of 0.01 to 0.70 which means that in a generation, probability of a one of the genes in a chromosome will mutate is 70%. The GA fitness function which evaluates the

fitness of a candidate design which is power dissipation in our case, we have first evaluated whether the CPI per thread for the chosen set of micro-architectural values is actually able to meet the real time requirements. The pseudo-code of our fitness, mutate gene and crossover two genes functions are described below. MAX_POWER is a maximum power level set to identify this design does not satisfy throughput constraints.

GA Fitness Function:

1. CPI_{Thread} = evaluate_thr_strand();
2. clk_period = (double) 1 / (double) CLK_FREQ;
3. if $CPI_{Thread} * InstrCnt_{PacketType} * clk_{period} \leq Packet\ Processing\ Time$ then
   return evaluate_power();
4. Else return MAX_POWER;

GA MutateGene Function:

1. Randomly choose a particular gene;
2. Randomly choose a new value for the gene within the range of the variable;
3. Set the value of the gene to the new value
4. Evaluate the CPI per thread;
5. If CPI per thread satisfies constraints, then accept new value
6. Else, find another new value for the gene;

GA 1pt Crossover Function:

1. Randomly choose a cut point for Gene1;
2. Crossover Gene1 and Gene2 from the cut point onwards;
3. Evaluate the CPI per thread for Gene1 and Gene2;
4. If CPI per thread satisfies constraints, then accept new values;
5. Else, find another new cut point;

GA 2pt Crossover Function:

1. Randomly choose 2 cut points for Gene1;
2. Crossover Gene1 and Gene2 from the cut point 1 till cut point 2;
3. Evaluate the CPI per thread for Gene1 and Gene2;
4. If CPI per thread satisfies constraints, then accept new values;
5. Else, find 2 new cut points;

Table 7-1 enlists the micro-architectural details of the best chromosomes or cores for the five packet types respectively found by GA. Rows 13a and 13b shows the model predicted and observed power dissipation values for the best candidate designs. Similarly, model predicted values and observed CPI per thread is shown in rows 14a and 14b. For each packet type, the 10 best cores from GA are stored and used later in the joint exploration of L2 memory and core micro-architectures. This completes the core optimization step.

Table 7-1: Optimized core architectures for five packet types found through GA

Parameters		**TYPE0**	**TYPE1**	**TYPE2**	**TYPE3**	**TYPE4**
Threads		16	16	16	16	4
LMQ		1	1	1	32	32
SB		1	1	4	32	32
I$ Size		1K	1K	1K	128K	4K
I$ Assoc.		8	4	16	4	4
I$ Linesize		64	64	16	128	16
D$ Size		1K	1K	4K	4K	128K
D$ Assoc.		16	16	8	4	4
D$ Linesize		128	2	64	32	64
MIL		32	32	32	1	1
ASIQ		1	4	32	8	16
Power (mW)	Pred.	202	212	228	297	261
	Obs.	230	229	227	310	265
CPI per thread	Pred.	19.4	20.1	21.7	20.19	6.99
	Obs.	18.9	18.7	19.8	21.72	7.13

To see the benefit Table 7-2 enlists the micro-architectural details of the best chromosomes or cores for the five packet types respectively found by GA. Rows 13a and 13b shows the model predicted and observed power dissipation

values for the best candidate designs.

Table 7-2: Comparison of power and CPI of optimized architectures found by GA with baseline architecture

Parameters	**Base Arch.**	**TYPE0**	**TYPE1**	**TYPE2**	**TYPE3**	**TYPE4**
Threads	4	16	16	16	16	4
LMQ	1	1	1	1	32	32
SB	8	1	1	4	32	32
I$ Size	32K	1K	1K	1K	128K	4K
I$ Assoc.	4	8	4	16	4	4
I$ Linesize	32	64	64	16	128	16
D$ Size	8K	1K	1K	4K	4K	128K
D$ Assoc.	4	16	16	8	4	4
D$ Linesize	16K	128	2	64	32	64
MIL	1	32	32	32	1	1
ASIQ	2	1	4	32	8	16
Power (mW)	261	Pred.	202	212	228	297
	265	Obs.	230	229	227	310
CPI per	6.99	Pred.	19.4	20.1	21.7	20.19

thread						
	7.13	Obs.	18.9	18.7	19.8	21.72

Step 3 (Integrated L2 and Core Optimization): The set of heterogeneous core architectures achieved above are used in the third and final stage where they are fed into a *Simulated Annealing*-based (SA) optimization engine [79, 80]. We have only considered one interconnection design which is a crossbar with fixed-sized queue.

$$N_{C_i} = NP_{TYPE_i} * \frac{T(N_{T_i})}{N_{T_i}} \qquad (7-1)$$

The number of required cores is calculated using Equation 7-1. For a packet type i if NP_i is the number of packets to be processed per second where $T(N_{T_i})$ is the processing time of that packet type in a core with N_T threads, total number of cores N_{C_i} is given by Equation 7-1.

The shared memory level L2 queue size, L2 size, associativity, line size and most importantly number of L2 banks, described in Table 7-3 are the five parameters which we explore to determine the overall L2 bandwidth, throughput and power dissipation of the entire chip. As the number of cores in a chip scales, the contention in the secondary cache increases resulting in non-deterministic L2 access times which exacerbates throughput degradation in the cores. This effect can be minimized by increasing the number of L2 banks. The L2 bank ids are typically decoded into the lower significant word of the physical addresses. This means that a fetched data block from main memory is distributed across all the

L2 banks according to the bank identification bits in the fetched address. This spatial distribution of data blocks across all the banks minimizes the number of simultaneous memory accesses per shared bank and attempts to mitigate the contention in each bank. However, as number of banks is scaled, L2 power dissipation increases as more logic is required to support the organization of the independent L2 banks. To address these design trade-offs, we perform a joint exploration of shared L2 and the cores such that sufficient data bandwidth can be provided to the cores and overall chip power dissipation can be minimized subject to the real time throughput demands of the NeP packet processing applications.

Table 7-3: Micro-architectural parameters of shared L2

Name	Range	Increment	Description
14. Q_{SIZE}	4 to 16	Power of 2	L2 input queue size per core
15. Size	4 to 512 MB	1MB	Total L2 size
16. Associativity	8 to 64	Power of 2	Set-associativity of L2 cache
17. Line Size	8 to 128	Power of 2	Line size of L2 cache
18. N_B	4 to 128	Power of 2	Number of L2 banks

The SA engine uses a L2 macro simulator called L2MacroSim which models the contention in the L2 cache in CASPER. Only the core to L2 cache

and L2 cache to memory reply/acknowledgement packets will be simulated. The inputs to the L2 MacroSim are L2 cache input queue size per core, cache bank size, line size, associativity, number of L2 banks, L1 I and D cache sizes, line sizes and associativities and instruction trace files for each thread in each core. The individual core parameters will be set to their optimal values from GA optimization. The L2MacroSim enables significant savings in simulation time while capturing the interaction between the cores. The macro-simulator also provides the power dissipation of the crossbar interconnection network and the L2 cache banks. The SA-based hill climbing algorithm is shown below:

```
 Define micro-architecture;
 Cost_fn_old = evaluate_power_dissipation();
 int inner_loop = 0, count = 0;


/* Initial Temperature */
 int T = T_0;
/* Initial Iteration */
 int iterations = I_0;
/* Repeat until Run-Time permits */
 while ( count++ < SA_COUNTER )
{
/* Repeat until inner loop iteration is not over */
        inner_loop = 0;
        while ( inner_loop++ < iterations )
```

```
        {
                /* Compute new Cost Funtion */
                //RF_count = 0;
                Cost_fn_new = perturb(L2 structure);

                // Hill-climbing part
                if ( Cost_fn_new > Cost_fn_old || ( rand() < e^((CostFnNew -CostFnOld)/T) ) )
                {
                        Cost_fn_old = Cost_fn_new;
                }
        }
/* Compute the new iteration for inner loop and Temperature */
        iterations = 1.2 * iterations;
        T = 0.1 * T;
}
```

The small hill-climbing technique embedded in the SA enables us to quickly converge to an optimal design thus giving us a shared memory heterogeneous many-core micro-architecture. The cost function in the SA is average power dissipation per cycle and constraints are the real-time throughput boundaries of the packets.

Figure 7-2 demonstrates how the number of threads per core changes during simulated annealing. The optimal number of threads per core for the cores

designed for different packet types is shown in Table 7-1. The high density of threads per core decreases single thread performance significantly to the extent that the threads processing packets cannot meet real-time constraints anymore. Hence, we observe that number of threads per core is scaled down to meet the performance requirements.

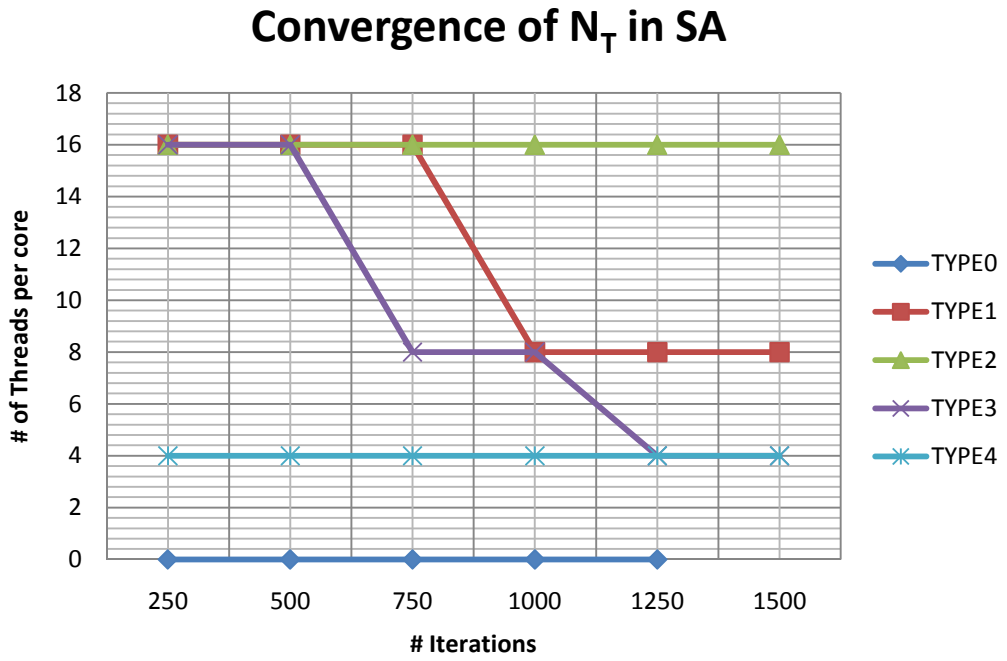

Figure 7-2: Thread scaling observed during simulated annealing

Table 7-4: Example optimal design found by simulated annealing

Micro-Architecture Specification	**Values**
N_C	214
N_C Types	5 (TYPE0, TYPE1, TYPE2, TYPE3, TYPE4)
NT per core (packet) type	8, 8, 8, 4, 4

N_C per packet type	80, 9, 19, 42, 65
N_B	32
L2 Size	256MB
L2 Bandwidth	36.352 GBps
Interconnect Bandwidth	32TBps
Average Estimated Power Dissipation	~80W
Effective Packet Bandwidth	329 GBps
Total Estimated Area	1930 mm^2

The result of SA-based optimization engine is shown in Table 7-4. A total number of cores = 214, where number of TYPE0 cores is 80, number of TYPE1 cores is 9, number of TYPE2 cores is 19, number of TYPE3 cores is 42 and number of TYPE4 cores is 65. 1 core was optimized for the DRR deficit round robin scheduling function. However, we have observed that a naïve deficit round robin scheduling will not suffice in such a large scale system. We believe that an out-of-order core will be able to exploit the instruction level parallelism of the scheduling algorithm and will perform better. Number of L2 banks used is 32 and the total L2 cache size is 256MB. The average power dissipation of the entire chip is around 80.9W and the net line speed achieved is 329 Gbps. The L2 cache memory section was able to provide a bandwidth of 36.352 GBps which was sufficient to keep the cores busy. The available bandwidth of crossbar interconnection network is 32TBps. The overall area of the chips is approximately 1930 mm^2.

Table 7-5: Comparison with other NePs

Specifications	**Netronome [4]**	**CISCO Quantum Flow [1]**	**Tilera [81]**	**UltraSPARC T1 [55]**	**Derived NeP**
#Cores	40	40	64	8	~200
#Threads	4	4	4	4	4 to 8
Power	-	400mW	-	-	80.9W
N/W Bandwidth	40Gbps	100+Gbps	40Gbps	-	329 Gbps
Heterogeneous	Yes	Yes	No	No	Yes

Table 7-5 shows the comparison of the derived NeP with other commercially available processors. Although our design space exploration method was able to achieve the highest throughput, number of cores is almost 5 times compared to the other NePs. Number of threads per core also varies in our case from 4 to 8. Number of threads per core in all other NePs is fixed. The power dissipation is significantly high compared to other NePs. The reason is significantly large number of cores, larger number of cache banks and a crossbar interconnection. Due to the fundamental differences with the other NePs available today we think direct comparison of our derived design is an unfair comparison.

CHAPTER 8: CONCLUSION

In this dissertation we have demonstrated an efficient scalable design space exploration framework for many-core heterogeneous embedded processors. In the current implementation of our framework, we have used a terabit per second network packet processing benchmark. In future we intend to explore a wide range of embedded applications where the different characteristics of various applications will pose different design challenges. We defined the core micro-architectural design space in terms of 13 parameters for each of the 5 IP packet types. Our objective was to use statistical machine learning to derive linear regression models of CPI per thread and power dissipation of the cores in terms of the micro-architectural design space parameters. The strength of this method is that even with a relatively fewer time-consuming cycle-accurate simulations (500-600), we were able to capture the complex relation of the performance and power dissipation of the cores within an error budget of 10%. However note that the proposed framework is flexible enough to explore various other machine learning and modeling techniques other than SML to study the power-performance trade-offs in embedded processor design. Our proposed method of pruning the design space by first optimizing core architectures using the derived linear regression models in the GA-based optimization engine and then integrating the L2 design parameters with the core architecture parameters in a SA-based hill-climbing algorithm enabled us to

rapidly achieve optimal power-performance point. Finally, we derived a many-core NeP with 214 cores and a 32-banked shared L2 cache which achieved a net line speed of 329 gigabits per second. We also found that the optimal number of hardware thread per core is 8. Scaling the number of hardware thread per core beyond 8 resulted in poor CPI per thread which failed to meet the real time constraints. In future, we also want to explore other exploration techniques such as neural networks and likewise to study whether better optimal design points can be achieved. Moreover, we were successful in avoiding simulation-in-loop methods as well as exhaustive search techniques which are extremely time-consuming and not cost-effective. Yet, our method produce results within a boundary of 20% error which we believe can be minimized by investing more time in collecting sample data set and fine tuning the linear regression models. In the end, even for such a large scale many-core system, we could keep the average power dissipation of the entire chip within 80W.

REFERENCES

[1] C. Inc. (2010). *The Cisco QuantumFlow Processor: Cisco's Next Generation Network Processor*. Available: http://www.cisco.com/en/US/prod/collateral/routers/ps9343/solution_overview_c22-448936.html

[2] Slovati. (2009, Dec 29, 2009). *The new Freescale QorlQ platform: how to migrate to an all-IP environment*.

[3] M. W. C. 2011, "NetLogic Microsystems Introduces the Industry's Most Advanced Multi-Core Processor for LTE eNodeB / Base Stations," 2011.

[4] N. Inc. (2010). *Netronome Heterogeneous Reference Architecture*. Available: http://www.netronome.com/pages/heterogeneous-architecture

[5] E. Lindholm*, et al.*, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *Micro, IEEE,* vol. 28, pp. 39-55, 2008.

[6] http://www.picochip.com/page/12/multi-core-dsp. (2010). *Multi-core DSP*.

[7] PressRelease. (2010, Annual Cisco Visual Networking Index Forecast Projects Global IP Traffic to Increase More Than Fourfold by 2014. Available: http://newsroom.cisco.com/dlls/2010/prod_060210.html

[8] L. G. Roberts, "A radical new router," *Spectrum, IEEE,* vol. 46, pp. 34-39, 2009.

[9] S. Timothy, "A Pipelined Memory Architecture for High Throughput Network Processors," 2003, pp. 288-288.

[10] T. Ungerer*, et al.*, "A survey of processors with explicit multithreading," *ACM Comput. Surv.,* vol. 35, pp. 29-63, 2003.

[11] M. A. M. Vieira*, et al.*, "Survey on wireless sensor network devices," in *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, 2003, pp. 537-544 vol.1.

[12] J. R. Allen*, et al.*, "IBM PowerNP network processor: Hardware, software, and applications," *IBM Journal of Research and Development,* vol. 47, pp. 177-193, 2003.

[13] T. Wolf, "Challenges and Applications for Network-Processor-Based Programmable Routers," in *Sarnoff Symposium, 2006 IEEE*, 2006, pp. 1-4.

[14] B. LILJEQVIST, "Visions and Facts – A Survey of Network Processors," Electrical Engineering Program, Department of Computer Engineering, CHALMERS UNIVERSITY OF TECHNOLOGY, Göteborg, 2003.

[15] A. C. Snoeren*, et al.*, "Single-packet IP traceback," *IEEE/ACM Trans. Netw.,* vol. 10, pp. 721-734, 2002.

[16] C. Fraleigh*, et al.*, "Packet-level traffic measurements from the Sprint IP backbone," *Network, IEEE,* vol. 17, pp. 6-16, 2003.

[17] R. Ramaswamy*, et al.*, "Analysis of network processing workloads," *J. Syst. Archit.,* vol. 55, pp. 421-433, 2009.

[18] C. Lewis and S. Pickavance, "Application/Bandwidth Requirements," in *Selecting MPLS VPN Services*, ed: Cisco Press, 2006, p. 456.

[19] C. Rosewarne, "Network Processors," Calyptech Inc.2004.

[20] http://www.ezchip.com/t_npu_whpaper.htm. (2010, NPU Designs for Next-Generation Networking Equipment.

[21] Intel. (2001, Next Generation Network Processor Technologies.

[22] L. Spracklen and S. G. Abraham, "Chip multithreading: opportunities and challenges," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, 2005, pp. 248-252.

[23] Virtutech, "Virtutech Simics."

[24] *Augmint*. Available: http://iacoma.cs.uiuc.edu/augmint.html

[25] *RSIM*. Available: http://rsim.cs.illinois.edu/rsim/dist.html

[26] *GEMS*. Available: http://www.cs.wisc.edu/gems/

[27] SimFlex. *SimFlex*. Available: http://si2.epfl.ch/~parsacom/projects/simflex/

[28] Z. Hui*, et al.*, "MPTLsim: A simulator for X86 multicore processors," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, 2009, pp. 226-231.

[29] M. T. Yourst, "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator," in *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, 2007, pp. 23-34.

[30] L. Yan*, et al.*, "NePSim: a network processor simulator with a power evaluation framework," *Micro, IEEE,* vol. 24, pp. 34-44, 2004.

[31] L. Sheng*, et al.*, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, 2009, pp. 469-480.

[32] P. J. Joseph*, et al.*, "A Predictive Performance Model for Superscalar Processors," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, 2006, pp. 161-170.

[33] C. Dubach*, et al.*, "Microarchitectural Design Space Exploration Using an Architecture-Centric Approach," presented at the Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, 2007.

[34] T. S. Karkhanis and J. E. Smith, "Automated design of application specific superscalar processors: an analytical approach," *SIGARCH Comput. Archit. News,* vol. 35, pp. 402-411, 2007.

[35] T. Sherwood*, et al.*, "Balancing design options with Sherpa," presented at the Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems, Washington DC, USA, 2004.

[36] B. C. Lee*, et al.*, "CPR: Composable performance regression for scalable multiprocessor models," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, 2008, pp. 270-281.

[37] E. *, et al.*, "Efficiently exploring architectural design spaces via predictive modeling," *SIGARCH Comput. Archit. News,* vol. 34, pp. 195-206, 2006.

[38] K. Sukhun and R. Kumar, "Magellan: A Search and Machine Learning-based Framework for Fast Multi-core Design Space Exploration and Optimization," in *Design, Automation and Test in Europe, 2008. DATE '08*, 2008, pp. 1432-1437.

[39] K. Sukhun and K. Rakesh, "Magellan: a search and machine learning-based framework for fast multi-core design space exploration and optimization," presented at the Proceedings of the conference on Design, automation and test in Europe, Munich, Germany, 2008.

[40] W. Tilman, "Performance Models for Network Processor Design," *IEEE Transactions on Parallel and Distributed Systems,* vol. 17, pp. 548-561, 2006.

[41] M. Shashidhar, "Exploring the Processor and ISA Design for Wireless Sensor Network Applications," 2008, pp. 59-64.

[42] Y.-N. Lin*, et al.*, "Modeling and analysis of core-centric network processors," *ACM Trans. Embed. Comput. Syst.,* vol. 8, pp. 1-15, 2009.

[43] E. S. Mostafa, "Architecture-Level Design Space Exploration of Super Scalar Microarchitecture for Network Applications," 2010, pp. 269-272.

[44] E. A. Brewer, "High-level optimization via automated statistical modeling," *SIGPLAN Not.,* vol. 30, pp. 80-91, 1995.

[45] R. Vuduc, "Automatic Performance Tuning of Sparse Matrix Kernels," University of California, Berkeley, 2003.

[46] J. Cavazos*, et al.*, "Rapidly Selecting Good Compiler Optimizations using Performance Counters," in *Code Generation and Optimization, 2007. CGO '07. International Symposium on*, 2007, pp. 185-197.

[47] A. Ganapathi*, et al.*, "A case for machine learning to optimize multicore performance," presented at the Proceedings of the First USENIX conference on Hot topics in parallelism, Berkeley, California, 2009.

[48] S.-w. Liao*, et al.*, "Machine learning-based prefetch optimization for data center applications," presented at the Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, Portland, Oregon, 2009.

[49] L. Jiangtian*, et al.*, "Machine learning based online performance prediction for runtime parallelization and task scheduling," in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, 2009, pp. 89-100.

[50] H. Leather*, et al.*, "Automatic Feature Generation for Machine Learning Based Optimizing Compilation," in *Code Generation and Optimization, 2009. CGO 2009. International Symposium on*, 2009, pp. 81-91.

[51] T. Wolf and M. Franklin, "CommBench-a telecommunications benchmark for network processors," in *Performance Analysis of Systems and*

Software, 2000. ISPASS. 2000 IEEE International Symposium on, 2000, pp. 154-162.

[52] A. Tee*, et al.* (2003, Implication of End-user QoS requirements on PHY & MAC. *802.20 WG Call for Contributions*.

[53] F. S. Foundation, GNU General Public License (GPL).

[54] A. S. Leon*, et al.*, "A Power-Efficient High-Throughput 32-Thread SPARC Processor," *Solid-State Circuits, IEEE Journal of,* vol. 42, pp. 7-16, 2007.

[55] A. S. Leon*, et al.*, "The UltraSPARC T1 Processor: CMT Reliability," in *Custom Integrated Circuits Conference, 2006. CICC '06. IEEE*, 2006, pp. 555-562.

[56] I. Sun Microsystems, "OpenSPARC T1 Micro-Archiecture Specification," ed, 2006.

[57] S. M. Inc., "UltraSPARC Architecture 2007, Privileged and Non-Privileged Instructions," ed, 2008.

[58] P. Kongetira*, et al.*, "Niagara: a 32-way multithreaded Sparc processor," *Micro, IEEE,* vol. 25, pp. 21-29, 2005.

[59] C. Dhruba*, et al.*, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, 2005, pp. 340-351.

[60] D. T. a. S. T. a. N. Jouppi, "CACTI 4.0," HP Laboratories, Palo AltoJune 2 2006.

[61] P. Subbarao*, et al.*, "Complexity-effective superscalar processors," vol. 25, ed: ACM, 1997, pp. 206-218.

[62] S. Inc., "DFT Compiler Datasheet," ed, 2009.

[63] Z. Wei and C. Yu, "New generation of predictive technology model for sub-45nm design exploration," in *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on*, 2006, pp. 6 pp.-590.

[64] *Cadence Encounter*. Available: http://www.cadence.com/products/ld/rtl_compiler/

[65] A. Gifi, *Nonlinear Multivariate Analysis*: John Wiley $ Sons, 1989.

[66] *SPECJBB2005 Benchmark Programs*. Available: http://www.spec.org/jbb2005/

[67] C. Ware, "The OSI network layer: Standards to cope with the real world," *Proceedings of the IEEE,* vol. 71, pp. 1384-1387, 1983.

[68] Canturk Isci*, et al.*, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, 2006, pp. 347-358.

[69] Kushal Datta and A. Mukherjee, "SPARC-based Cycle-Accurate CMT Architecture Simulator for Performance Energy and Area Analysis," University of North Carolina at Charlotte, Charlotte2008.

[70] G. E. Tellez*, et al.*, "Activity-driven clock design for low power circuits," in *Computer-Aided Design, 1995. ICCAD-95. Digest of Technical Papers., 1995 IEEE/ACM International Conference on*, 1995, pp. 62-65.

[71] S. Mutoh*, et al.*, "1-V power supply high-speed digital circuit technology with multithreshold-voltage CMOS," *Solid-State Circuits, IEEE Journal of,* vol. 30, pp. 847-854, 1995.

[72] P. Macken*, et al.*, "A voltage reduction technique for digital systems," in *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International*, 1990, pp. 238-239.

[73] Arindam Mukherjee*, et al.*, "Chapter 10: Hardware Techniques for Autonomous Power Saving in Embedded Many-Core Processors," in *Multi-Core Embedded Systems*, G. Kornaros, Ed., 1 ed: CRC Press and Taylor & Francis Group, 2009.

[74] Kim Wonyoung*, et al.*, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *High Performance Computer Architecture, 2008. IEEE 14th International Symposium on*, 2008, pp. 123-134.

[75] Alon Naveh*, et al.*, "Power and Thermal Management in The Intel Core Duo Processor," *Intel Technology Journal,* vol. 10, May 15 2006.

[76] Luca Benini and G. D. Micheli, *Dynamic power management: Design Techniques and CAD Tools*: Kluwer Academic, 1997.

[77] A. Presta. (2007). *Fast Genetic Algorithm*.

[78] B. Jong-Ho*, et al.*, "Performance analysis of coarse-grained parallel genetic algorithms on the multi-core sun UltraSPARC T1," in *Southeastcon, 2009. SOUTHEASTCON '09. IEEE*, 2009, pp. 301-306.

[79] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *Journal of Statistical Physics,* vol. 34, pp. 975-986, 1984.

[80] K. Datta*, et al.*, "Automated design flow for diode-based nanofabrics," *J. Emerg. Technol. Comput. Syst.,* vol. 2, pp. 219-241, 2006.

[81] S. Bell*, et al.*, "TILE64 - Processor: A 64-Core SoC with Mesh Interconnect," in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, 2008, pp. 88-598.