STATISTICAL MACHINE LEARNING BASED MODELING FRAMEWORK FOR DESIGN SPACE EXPLORATION AND RUN-TIME CROSS-STACK ENERGY OPTIMIZATION FOR MANY-CORE PROCESSORS

by

Changshu Zhang

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 Engineering

Charlotte

2013

Approved by:

____
Dr. Arun Ravindran

____
Dr. Arindam Mukherjee

____
Dr. Bharat S. Joshi

____
Dr. Yuanan Diao

ABSTRACT

CHANGSHU ZHANG. Statistical machine learning based modeling framework for design space exploration and run-time cross-stack energy optimization for many-core processors. (Under the direction of DR. ARUN RAVINDRAN)

The complexity of many-core processors continues to grow as a larger number of heterogeneous cores are integrated on a single chip. Such systems-on-chip contains computing structures ranging from complex out-of-order cores, simple in-order cores, digital signal processors (DSPs), graphic processing units (GPUs), application specific processors, hardware accelerators, I/O subsystems, network-on-chip interconnects, and large caches arranged in complex hierarchies. While the industry focus is on putting higher number of cores on a single chip, the key challenge is to optimally architect these many-core processors such that performance, energy and area constraints are satisfied. The traditional approach to processor design through extensive cycle accurate simulations are ill-suited for designing many-core processors due to the large microarchitecture design space that must be explored. Additionally it is hard to optimize such complex processors and the applications that run on them statically at design time such that performance and energy constraints are met under dynamically changing operating conditions.

The dissertation establishes statistical machine learning based modeling framework that enables the efficient design and operation of many-core processors that meets performance, energy and area constraints. We apply the proposed framework to rapidly design the microarchitecture of a many-core processor for multimedia, computer graphics rendering, finance, and data mining applications derived from the Parsec benchmark. We further demonstrate the application of the framework in the joint run-time adaptation of both the application and microarchitecture such that energy availability constraints are met.

ACKNOWLEDGEMENTS

First and foremost, I want to thank my advisor, Dr. Arun Ravindran. His advice gives me the inspiration to continuously make progress on my research. I am particularly grateful for his patience with me, his generous help and great encouragement. I also feel grateful to my committee members, Dr. Arindam Mukherjee, Dr. Bharat S. Joshi, and Dr. Yuanan Diao for reviewing my work and providing me very helpful suggestions.

I would like to thank my parents Mr. Jun Zhang and Ms. Yanyan Yin for their love, their unconditional support, and for being so proud of me. I can’t thank more to my wife Ms. Zhengzheng Yu for being my best friend and for giving me so much. I feel so fortunate to have her firmly by my side in going through times of toughness and happiness in life together. Additionally, I would like to thank the rest of my family, my aunts and uncles, and cousins for always being there for me whenever I need help.

I would like to give big thanks to my dear friends Guangyi Cao, Kushal Datta and Jong-ho Byun, who give me tremendous advice and support.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

The complexity of many-core processors continues to grow as a larger number of heterogeneous cores are integrated on a single chip. Such systems-on-chip contains computing structures ranging from complex out-of-order cores, simple in-order cores, digital signal processors (DSPs), graphic processing units (GPUs), application specific processors, hardware accelerators, I/O subsystems, network-on-chip interconnects, and large caches arranged in complex hierarchies. While the industry focus is on putting higher number of cores on a single chip, the key challenge is to optimally architect these many-core processors such that performance, energy and area constraints are satisfied. The traditional approach to processor design through extensive cycle accurate simulations are ill-suited for designing many-core processors due to the large microarchitecture design space that must be explored. Additionally it is hard to optimize such complex processors and the applications that run on them statically at design time such that performance and energy constraints are met under dynamically changing operating conditions.

The goal of the dissertation is to establish statistical machine learning (SML) based modeling framework that enables the efficient design and operation of many-core processors that meets performance, energy and area constraints. The dissertation makes the following contributions in this regard -

1. Determine the performance of different classes of SML algorithms in modeling the performance, power, and area of a many-core processor.

2. Establish a hierarchical modeling methodology that addresses the complexities of modeling many-core processors by combining individual models of the different micro-architectural elements to create a many-core performance, power, and area model.
3. Establish a SML based modeling and Pareto optimal exploration framework for run-time cross-stack energy optimization.
4. Demonstrate the performance vs. quality tradeoffs possibilities for the x264 video encoder through by tuning the motion estimation algorithm and the video frame resolution.
5. Demonstrate the use of feature reduction algorithms in evaluating the impact of micro-architectural parameters on performance, power, and area.
6. Develop application-specific-performance versus power relationships for several applications derived from the Parsec benchmark.

1.1 Motivation

A many-core processor architect has many degrees of freedom in choosing the values of micro-architectural parameters that meets processing, power dissipation, and area constraints. Since each of these micro-architectural parameters can assume a range of possible values, the resulting design space consists of millions design points. Exploring this vast design space to find optimal power-performance configurations, with the traditional method of cycle-accurate simulation to estimate performance followed by register-transfer-level (RTL) synthesis to estimate power, is impractical because of the large simulation time (tens of hours) needed for each design point. Additionally, contention between the cores for shared resources such as interconnection networks, caches etc., and results in super-linear increase in simulation time as the number of cores is increased. Recently, statistical methods have been proposed both for uniprocessor and many-cores, that seek to develop computationally efficient substitutes for exhaustive cycle accurate simulation. These methods sample a fraction of the design space to generate training data that map architecture design parameters to performance metrics. However, dependences among the design parameters results in complex non-linear relationships between these parameters and the performance metrics making the modeling effort challenging.

The complexity of many-core processor continues to grow as a larger number of heterogeneous cores are integrated on a single chip. Often it is hard to optimize such complex cores at design time so that performance requirements are met along with temperature, energy, noise, and reliability constraints. Dynamic performance optimization thus become a necessity - where the operational points comprising of all parts of the computing stack - the architecture, system software, and applications are adjusted at run-time to meet performance requirements and operating constraints. As an example, for battery based mobile computing devices, the amount of energy available for computing is fixed between charges. As a result, the energy available for computing changes dynamically depending on the number of tasks running on the system, which may be hard to predict in advance. Note that due to complex interactions between the different parts of the stack, optimal performance requires that the operating points across the computing stack must be tuned simultaneously. Also, for applications with hard performance constraints such as real time requirements, separately optimizing the architecture, operating system, and application quality-of-service operating parameters may result in a scenario where no operating point can satisfy performance constraints.

Previous research has identified a number of architecture parameters whose operating values can be tuned during run-time - core parameters such as voltage/frequency, the number of integer and floating point units, integer and floating point register sizes, instruction and store queue sizes; cache and TLB parameters - capacity, line size, and associativity; interconnect parameters - number of channels, flit size, and channel width. Among the system software parameters that can be tuned include CPU allocation and budgets for real time scheduling algorithms. Application parameters that can be tuned depend on user preferences and a given application. For many soft real time applications such as video conferencing, the user may be willing to tradeoff video frame size and visual quality for battery life if throughput requirements dictated by the video standard are met. Since each parameter of the computing stack can take on a number of operational values, the operational space can consist of tens of millions of points that must be evaluated at run-time to determine the optimal operating point. The cross-stack optimization approaches reported in literature thus far have considered a very limited space of operational points.

1.2 Contributions

1.2.1 Comparison of SML Models

Motivated by the success of SML algorithms in modeling complex systems, in this dissertation we systematically investigate and compare the performance of different classes of SML algorithms in modeling the power and performance of a multi-core processor. Although previous research has used specific SML algorithms for modeling the performance of multi-core processors [1] [2], no systematic study comparing the relative merit of these algorithms has been reported in the literature. The SML algorithms we consider are Multivariate Adaptive Regression Splines (MARS), Kernel Canonical Correlation Analysis (KCCA), Artificial Neural Networks (ANN), and Support Vector Machines for Regression (SVM-R). These algorithms represent different trade-offs between training time and modeling accuracy. Our results indicate that despite training on a very small set of samples (< 0.001%), a high prediction accuracy is possible (> 90%) with MARS for diverse applications derived from the Parsec benchmark [3]. We then utilize the predictive MARS model for micro-architectural design space exploration and use evolutionary algorithms to generate Pareto optimal power-performance fronts.

1.2.2 Hierarchical Scalable Modeling Framework

Despite the vast reduction in design time possible with the SML model driven micro-architectural exploration, its scalability as the number of cores increase is limited. The primary reason for this limited scalability is the super-linear increase in simulation time required to generate the training data set. In this dissertation, we therefore, propose a hierarchical SML modeling methodology that consistently combines the statistical models of cluster of individual micro-architectural elements to generate an overall performance, power, and area model for a many-core processor with potentially hundreds of processing cores. The entire micro-architectural design space of the many-core processors is divided into core-level parameters and interconnection network-level parameters. Through the simulation of the individual cores with core-level parameters, the performance, power, and injection rate models are constructed. Similarly, through the simulation of the interconnection network with network-level parameters, the interconnection network average latency model is constructed. The injection rate affects the interconnection network latency, while the interconnection network latency reacts on the injection rate.

We then find the convergence of these two models and use the corresponding interconnection network latency to obtain the performance and power consumption of the individual cores.

We apply the hierarchical modeling framework to the Parsec benchmark. The target architecture is a 64-core many-core processor with the cores based on the Alpha 21264 out-of-order processor. The cores have private L1 data and instruction cache, and four cores are clustered sharing a unified cache coherent L2 cache through a bus based interconnect. The interconnect is a 4x4 directory based mesh network with 16 memory controllers. Each cluster is an individual network node in the interconnect. We simulate the performance of the individual cores using the Gem5 architectural simulator, the performance of the interconnect using the GARNET network simulator and estimate the entire many-core processor power using the McPAT modeling framework targeting the 22nm technology. Our results indicate that similarly to the cluster level performance, power and area modeling, despite training on a small set of samples (< 10%), a high prediction accuracy is achieved (> 90%) with MARS for the cluster injection rate, interconnect access latency, power and area modeling. We then iterate the cluster injection rate and interconnect access latency MARS models till these two models converge and consistent. For the benchmark selected in our work, the cluster injection rate model and interconnect access latency model converge very fast, normally within tens of iterations. The many-core model is thus a combination of the cluster performance, power, area models and interconnect power, area models. Evaluations of our many-core models constructed using the MARS algorithms, shows an R^2 ranging from 0.82 – 0.89 for selected Parsec benchmarks with the simulation time increasing linearly in the worst-case with the number of cores.

1.2.3 Run-time Cross-stack Performance-energy Optimization

As mentioned before, performance and power optimization at design time cannot meet energy availability constraints that are only apparent at run-time. Hence we apply the SML based modeling framework to dynamically optimize performance and power by jointly tuning both application and micro-architectural parameters at run-time. From a training set composed of a small fraction of operational points (< 1%), we construct a MARS model with both micro-architectural and application parameters as predictor variables. The model predicts the power and performance of the operating points outside the training set. We employ a feature reduction algorithm to identify the parameters that most significantly contributes to performance and power, thus reducing the operating space to be explored. Since we seek to optimize multiple objectives in an unbiased fashion, we use a Pareto front exploring evolutionary algorithm that uses the MARS model to determine operating points for optimal power and performance. The operating points constituting the Pareto front can be stored in look-up tables for rapidly determining the optimal operating point.

We apply the proposed framework to an x264 video encoding application obtained from the Parsec benchmark. The target architecture is a quad core processor with the cores based on the Alpha 21264 out-of-order processor. The cores have private L1 data and instruction cache, and a unified cache coherent L2 cache shared through a bus based interconnect. We simulate the performance using the Gem5 architectural simulator and estimate power using the McPAT modeling framework targeting the 22nm technology node. Note that the architecture is not specifically designed for video processing but rather chosen to illustrate our modeling and exploration framework. The micro-architectural predictor variables include a total of 10 core and cache parameters. The application predictor variables include the video resolution, and visual quality determined by the choice of the motion estimation algorithm. The model outputs the average frames per second (FPS) and the average power consumption. The MARS model has an R^2 of 0.9657 and 0.9467 and RMSE (root mean squared error) of 1.829 and 0.0124 respectively for FPS and power consumption. Comparison of the power consumption of Pareto optimal operating point at a lower visual quality to that of Pareto optimal point at a higher visual quality for an x264 video encoder executing on a prototype quad core processor indicates a power saving of 55%.

1.3 Organization

The remainder of the dissertation is organized as follows. Chapter 2 provides a comprehensive overview of SML algorithms considered in this work and a comparison of the performance of different SML algorithms in modeling the power and performance of a many-core processor. Chapter 3 describes the proposed scalable framework that can be used to generate SML performance, power, and area models of many-core processors with hundreds of heterogeneous processing cores. Chapter 4 discusses the extension of the proposed SML modeling framework in the run-time performance-energy optimization through a cross-stack approach. Chapter 5 concludes with a summary of the work presented in this dissertation and with the discussion of how future work could extend the results presented in this dissertation. All chapters are relatively self-contained with the necessary background and related work.

CHAPTER 2: A COMPARISON OF STATISTICAL MACHINE LEARNING ALGORITHMS IN MULTI-CORE DESIGN SPACE MODELING

2.1 Introduction

A multi-core processor architect has many degrees of freedom in choosing the values of micro-architectural parameters that meets processing and power dissipation constraints. Since each of these micro-architectural parameters can assume a range of possible values, the resulting design space consists of millions design points. Exploring this vast design space to find optimal power-performance configurations, with the traditional method of cycle-accurate simulation to estimate performance followed by register-transfer-level (RTL) synthesis to estimate power, is impractical because of the large simulation time (tens of hours) needed for each design point. In this chapter, we explore the statistical modeling of processors as an approach to make the exploration problem computationally tractable. In general, modeling of multi-core power and performance is not straightforward due to their non-linear dependence on processor parameters, and the interdependencies between the design parameters with each other.

Recent years have witnessed the development of powerful statistical machine learning (SML) algorithms that treat the system as a black-box but are capable of extraction of relationships between input parameters and performance metrics with good accuracy. Such SML algorithms have been successfully used in system modeling in fields as diverse as bioinformatics [4], communications [5], information management [6], and finance [7].

Motivated by the success of SML algorithms in modeling complex systems, in this chapter we systematically investigate and compare the performance of different classes of SML algorithms in modeling the power and performance of a multi-core processor. Although previous research has used specific SML algorithms for modeling the performance of multi-core processors [1] [2], no systematic study comparing the relative merit of these algorithms has been reported in the literature. The SML algorithms we consider are Multivariate Adaptive Regression Splines (MARS), Kernel Canonical Correlation Analysis (KCCA), Artificial Neural Networks (ANN), and Support Vector Machines for Regression (SVM-R). These algorithms represent different trade-offs between training time and modeling accuracy. Our results indicate that despite training on a very small set of samples (< 0.001%), a high prediction accuracy is possible (> 90%) with MARS for diverse applications derived from the Parsec benchmark [3]. We then utilize the predictive MARS model for micro-architectural design space exploration and use evolutionary algorithms to generate Pareto optimal power-performance fronts.

The rest of the chapter is organized as follows – in Section 2.2 we introduce the SML modeling framework. We review the SML algorithms considered in our work in Section 2.3. We describe the evaluation methodology in Section 2.4. We then present the results comparing the performance of different SML algorithm in Section 2.5. In Section 2.6, we describe the application of the SML model in design space exploration. We review related work in Section 2.7 and conclude the chapter in Section 2.8.

2.2 Machine Learning Framework

A statistical machine learning (SML) based approach to architecture modeling seeks to develop a SML based regression model that predicts architecture performance metrics (target variables) as a function of architectural design parameters (predictor variables) such that the model best fits the observed data and any prior knowledge held by the learner [8].

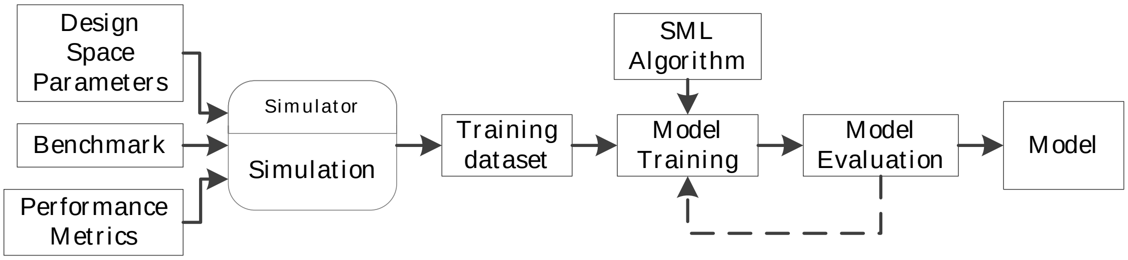

Figure 2.1: SML modeling framework

The general SML modeling framework is shown in Figure 2.1. We initially select architectural parameters (both core level and cache level) that can potentially affect the performance metrics of the architecture. Examples of such architectural parameters include number of cores, threads per core, cache sizes and associativity, buffer sizes, and so on. Examples of performance metrics include cycles-per-instruction (CPI), task throughput, and power dissipation. Note that each architectural design parameter can assume a range of possible values, the combinations of which can potentially result in a large design space. We then select a small subset of these combinations that adequately represents the design space. For this subset and a given application benchmark, we collect the corresponding architectural performance metrics using architecture simulators. We then train the SML model and evaluate the accuracy of the model through a suitable validation methodology. We iterate this process until a desired model accuracy is obtained.

2.3 Statistical Machine Learning Algorithms

In SML regression modeling, given a training data set of predictor variables and the corresponding target variables, the L target variables $\{\boldsymbol{y}\}$: $\{y_1, \dots, y_L\}$ are related to the P predictor variables $\{\boldsymbol{x}\}$: $\{x_1, \dots x_P\}$ through a model described by:

$$\boldsymbol{y} = \hat{f}(\boldsymbol{X}) + \boldsymbol{\epsilon} \tag{2.1}$$

Here $\boldsymbol{y}$ is a $L \times 1$ column vector of target variables, $\boldsymbol{X}$ is a $L \times P$ matrix of predictor variables such that each row has the same $1 \times P$ row vector $\boldsymbol{x}$. The transformation $\hat{f}$ captures the relationship between $\boldsymbol{X}$ and $\boldsymbol{y}$, and $\boldsymbol{\epsilon}$ is a $L \times 1$ error column vector which reflects error due to the effect of hidden predictor variables other than $\boldsymbol{x}$ on $\boldsymbol{y}$. In the following sections, we review the different SML algorithms considered in our work which we use to determine $\hat{f}$.

2.3.1 Multivariate Adaptive Regression Splines (MARS)

2.3.1.1 Linear regression

Linear regression is the most widely used model to construct the approximation function $\hat{f}$ for each target variable using a linear relationship:

$$y = \hat{f}(\boldsymbol{x}) + \epsilon = b + \sum_{i=1}^{P} \omega_i x_i + \epsilon \tag{2.2}$$

Here y is the target variable of interest, b is an intercept term, and ω_i is the corresponding coefficient of predictor variable x_i. A common measure of accuracy (loss function) is least-square fitting by minimizing the Mean Squared Error (MSE) of the model:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left[y_i - \hat{f}(\boldsymbol{x}_i)\right]^2 \tag{2.3}$$

Here N is the number of training samples.

2.3.1.2 Polynomial regression

To approximate a non-linear relationship between target variable and predictor variables, a Q^{th} order polynomial of predictor variables can be used yielding the following polynomial regression model:

$$\hat{f}(\boldsymbol{x}) = b + \sum_{m=1}^{P} \beta_m \prod_{i=1}^{m} {x_i}^{q_i} + \sum_{n=1}^{P} \omega_n \prod_{j=1}^{n} {x_j}^{q_j} \tag{2.4}$$

Here $\sum_{i=1}^{m} q_i < Q$, $\sum_{j=1}^{n} q_j = Q$, β_m and ω_n are coefficients. Similar to linear regression, the model is fitted by minimizing the MSE.

2.3.1.3 Regression spline

If a single polynomial fitting of the target variable and predictor variables is not accurate enough, a regression spline composed of two or more polynomial fittings is used. In univariate regression spline ($P = 1$), the range of the predictor variable *x* is divided into $K + 1$ disjoint regions separated by K points (knots). In each region, a Qth order polynomial regression is applied, and at each knot a continuous constraint is placed. Thus a total of $(K + 1)(Q + 1) - KQ$ coefficients have to be adjusted to best fit the training data by minimizing the MSE. Usually, this constrained minimization problem is solved by converting to an equivalent unconstrained optimization problem. The approximation function in each of the $K + 1$ region is chosen from a set of Qth order basis functions $\{B_k^Q(x)\}_{k=0}^{K+Q}$ and the univariate regression spline model is described by:

$$\hat{f}(x) = \sum_{k=0}^{K+Q} \omega_k B_k^Q(x) \tag{2.5}$$

One such basis is truncated power basis which consists of following functions:

$$\{x^j\}_{j=0}^{Q}, \{\pm\{x - t_k\}_{+}^{Q}\}_{k=1}^{K} \tag{2.6}$$

Here $\{t_k\}_{k=1}^{K}$ are the knot locations. The truncated power function pair is defined by

$$+(x - t_k)_+^Q = \begin{cases} 0 & x \leq t_k \\ (x - t_k)^Q & \text{otherwise} \end{cases} \tag{2.7}$$

$$-(x - t_k)_+^Q = \begin{cases} (t_k - x)^Q & x \leq t_k \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

Other basis functions could be used as well, such as B-spline basis [9]. The model in Equation 2.5 can then be fitted by minimizing the MSE. However, due to the great flexibility of regression spline in specifying the number of knots K and their locations $\{t_k\}_{k=1}^{K}$, the optimal specification for the knots is usually unknown. The number of knots is usually selected based on the distribution of the training data and the standard knot selection scheme includes either equal-spaced placing or at the $1/K(* 100)$ percentiles of the range of x.

Adaptive knot selection

Smith [10] suggested an adaptive knot selection strategy for automatically selecting both the number and locations for the knots. She used the truncated power basis; Equation 2.5 can then be written as

$$\hat{f}(x) = \sum_{q=0}^{Q} \beta_q x^q + \sum_{k=1}^{K} \omega_k [\pm(x - t_k)_+^Q] \tag{2.9}$$

Here $\{\beta_q\}_{q=0}^{Q}$ and $\{\omega_k\}_{k=1}^{K}$ are coefficients of $\{x^q\}_{q=0}^{Q}$ and $\{\pm(x - t_k)_+^Q\}_{k=1}^{K}$. For N training data, Smith's strategy involves starting from a very large number of eligible knot locations, such as $K = N - 2$. Then the number of knots is dropped progressively, until a Sum of Squared Error (SSE) threshold is exceeded.

2.3.1.4 Multivariate adaptive regression splines (MARS)

In multivariate regression spline, placing K_j knots for each predictor variable $\{x^q\}_{j=1}^{P}$ produces $\prod_{j=1}^{P}(K_j + 1)$ regions. The basis function set over this set of regions is the tensor product [11] of the corresponding univariate spline basis associated with the knot locations on each predictor variable:

$$\hat{f}(\boldsymbol{x}) = \sum_{k_1=0}^{K_1+Q} \dots \sum_{k_P=0}^{K_P+Q} \omega_{k_1}, \dots, \omega_{k_P} \prod_{j=1}^{P} B_{k_j}^{Q}(x_j) \qquad (2.10)$$

In Equation 2.10, there are $\prod_{j=1}^{P}(K_j + Q + 1)$ coefficients to be estimated. This is a reflection of “curse of dimensionality” [12] [13] as the number of coefficients has an exponential dependence on the number P and order Q of the predictor variables. The MARS strategy employs the truncated power basis of Equation s 2.7 and 2.8, adaptive knot selection on each predictor variable as suggested by Smith, and the tensor product splines of Equation 2.10. However, this strategy is computationally challenging. Adaptive knot selection scheme in multivariate case, involves computationally expensive $O(N^P)$ MSE fits. Such a scheme is only practical for small number of training data and predictor variables.

MARS algorithm

The goal of the following MARS algorithm [14] is to provide a computationally feasible approach that approximates the basis function subset selection procedure. One representation of these basis functions is:

$$B_m(\boldsymbol{x}) = \prod_{i=1}^{I_m} [s_{(i,m)}(x_{j_{(i,m)}} - t_{(i,m)})]_+^Q \qquad (2.11)$$

Here I_m is the number of factors (interaction order) of the m^{th} basis function and (i, m) denotes the m^{th} basis function and the i^{th} interaction order. $s_{(i,m)} = \pm 1$ indicates the positive (right) or negative (left) of the truncated power function pairs of the mth basis function. $x_{j_{(i,m)}}$ is one of the predictor variable $\{x_j\}_{j=1}^{P}$ and $t_{(i,m)}$ is the knot location.

The MARS algorithm employs a two-step procedure to build the model: the forward pass and backward pass. In the forward pass, the algorithm starts with one basis function which is just an intercept term: $B_0(\boldsymbol{x}) = 1$, and then repeatedly adds new basis functions in pairs (two at a time) to the model. The $(M+1)^{th}$ iteration adds following two new basis functions:

$$\begin{cases} B_{(2M+1)}(\boldsymbol{x}) = B_{l_{(M+1)}}(\boldsymbol{x})[+(x_{j_{(M+1)}} - t_{M+1})]_+^Q \\ B_{(2M+2)}(\boldsymbol{x}) = B_{l_{(M+1)}}(\boldsymbol{x})[-(x_{j_{(M+1)}} - t_{M+1})]_+^Q \end{cases} \tag{2.12}$$

Here $B_{l_{(M+1)}}(\boldsymbol{x})$ is chosen using Equation 2.11 from one of the $\{B_m(\boldsymbol{x})\}_{m=0}^{2M}$ generated in the previous M iterations. Note that if $l_{(M+1)} = 0$, then $B_0(\boldsymbol{x}) = 1$ is chosen, and the two newly added basis functions have only one truncated power basis function. In this case, the interaction order of $B_{(2M+1)}(\boldsymbol{x})$ and $B_{(2M+2)}(\boldsymbol{x})$ is 1. If $l_{(M+1)} > 0$, then a non-constant basis function $B_l(\boldsymbol{x})(1 \leq l \leq 2M)$ is chosen, and the two newly added basis functions have one higher interaction order than $B_l(\boldsymbol{x})$. The knot location in $(M+1)^{th}$ iteration on corresponding variable $x_{j_{(M+1)}}$ is given by t_{M+1}. The values of $l_{(M+1)}, j_{(M+1)}$ and t_{M+1} are obtained by maximizing the reduction in SSE (argmin) of the model built in the previous M iterations, and described by

$$\left(l_{(M+1)}, j_{(M+1)}, t_{M+1}\right) = \mathrm{argmin}_{l,j,t,(\omega_m)_0^{2M+2}} \sum_{i=1}^{N} \Big\{ y_i - \sum_{m=0}^{2M} \omega_m B_m(\boldsymbol{x}_i) - \omega_{2M+1} B_l(\boldsymbol{x}_i)\left[+\left(x_j - t\right)\right]_+^Q - \omega_{2M+1} B_l(\boldsymbol{x}_i)\left[-\left(x_j - t\right)\right]_+^Q \Big\}^2 \tag{2.13}$$

Here $\sum_{m=0}^{2M} \omega_m B_m(\boldsymbol{x}_i)$ is the model built in the previous M iterations and the latter two terms are the two newly added basis functions in the $(M+1)^{th}$ iteration. In Equation 2.13, all possible choices of l, j and t are evaluated until the change of SSE of the model is below a given threshold or until the maximum number M_{max} of basis functions is reached. M_{max} is typically chosen to be substantially larger than would be optimal, and results in an over-fitted model at the end of the forward pass.

In the backward pass, the MARS algorithm prunes the model built in the forward pass by removing the least effective basis functions one at a time until only the intercept term is left. The least effective basis function is determined based on a modification of the Generalized Cross Validation (GCV) criterion [15] as described by:

$$GCV(M) = \frac{1}{N}\sum_{i=1}^{N}[y_i - \hat{f}_M(\boldsymbol{x}_i)]^2/(1 - \frac{enp(M)}{N})^2 \quad (2.14)$$

$$enp(M) = M + c * \frac{M-1}{2} \quad (2.15)$$

Here M is the number of basis functions in the model $\hat{f}_M(\boldsymbol{x})$ (including the intercept term), enp is the effective number of parameters, c is the GCV penalty per knot and $(M-1)/2$ is the number of knots. The final output model is the one with the lowest GCV value. The numerator of Equation 2.14 is the MSE on the training data and the denominator represents a penalty for increasing model complexity (number of knots).

2.3.2 Artificial Neural Networks (ANN)

ANN is a non-linear statistical data model inspired by biological nervous systems and neural networks. The representational power of ANN is rich enough to express complex interactions among variables -- any function can be approximated to arbitrary precision by a three-layer ANN [16].

Three types of parameters define an ANN: the interconnection pattern, the learning process, and the activation function [16]. Figure 2.2 shows a feed-forward ANN and a sigmoid activation function. The feed forward network contains three layers with P units in the input layer, H units in the hidden layer, and L units in the output layer. The sigmoid activation function is applied to the units in the hidden layer and output layer. ω_{hij} represents the weight between input unit $\{x_i\}_{i=1}^{P}$ and hidden unit $\{h_j\}_{j=1}^{H}$, and ω_{ojk} represents the weight between hidden unit $\{h_j\}_{j=1}^{H}$ and output unit $\{y_k\}_{k=1}^{L}$. To train the network, we choose the stochastic gradient descent version of the BackPropagation (BP) algorithm [16]. The BP algorithm is the most commonly used ANN learning technique. It is composed of three parts: forward propagation, back propagation and weight update. In each iteration, the training input is first forwarded through the network to generate the output of each layer using the sigmoid function. Then, the errors are propagated backward through the network to calculate the error term of each unit. The weights in each layer are then updated using the gradient descent algorithm. This training process stops when the error difference between adjacent iterations is below a desired error threshold or a maximum number of iterations are reached.

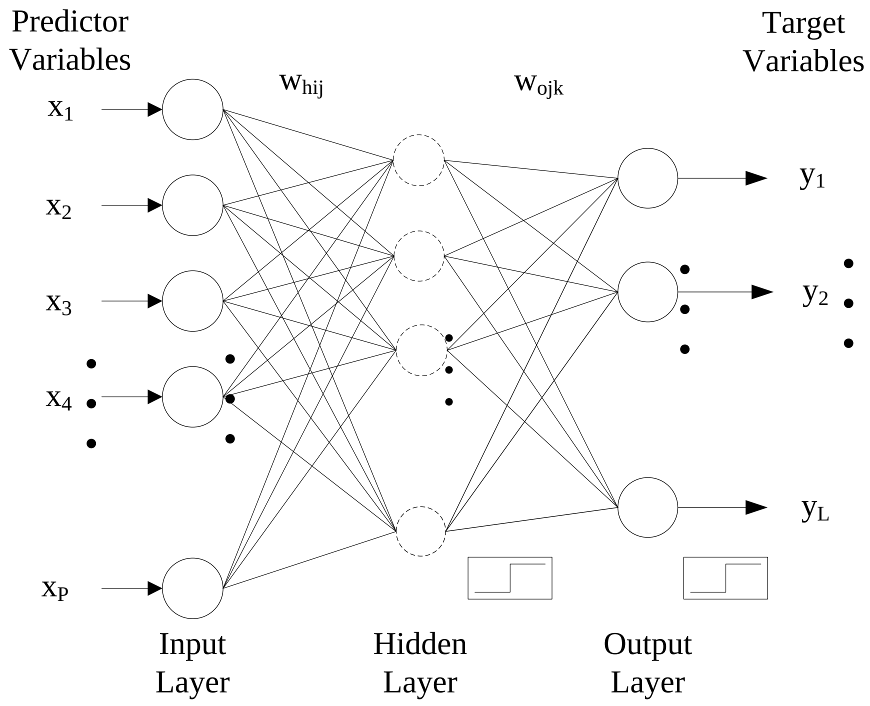

Figure 2.2: Example of a feed-forward ANN [16]

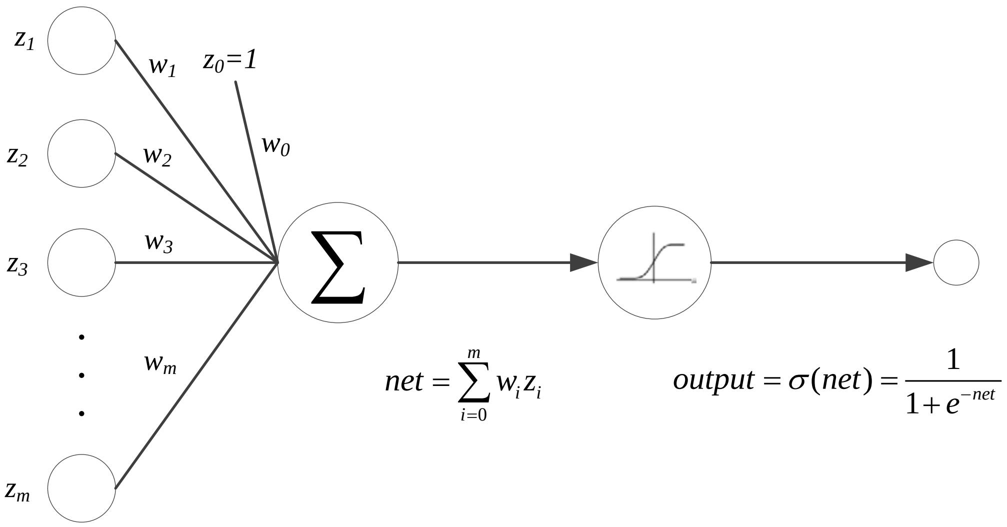
z_1
w_1
$z_0=1$
w_0
z_2
w_2
z_3
w_3
w_m
z_m
$\sum$
$net = \sum_{i=0}^{m} w_i z_i$
$output = \sigma(net) = \frac{1}{1+e^{-net}}$

Figure 2.3: Example of a sigmoid activation function [16]

Gradient descent is an optimization algorithm providing a basis for learning algorithms searching through a hypothesis space to find an optimal $\hat{f}$. In standard gradient descent, this hypothesis space H is normally composed of sum of squared errors (SSE) of all the training data with different weights. The error E is described by:

$$E(\boldsymbol{\omega}) = \frac{1}{2}\sum_{d\in D}(y_d - \hat{f}(\boldsymbol{x}_d))^2 \tag{2.16}$$

Here D is all the training data. For each training data d in D, y_d is the true target value, and $\hat{f}(\boldsymbol{x}_d)$ is the estimated output of the target. Given an ANN shown in Figure 2.3, the gradient of E is defined as:

$$\nabla E(\boldsymbol{\omega}) = [\frac{\partial E}{\partial \omega_{h11}}, \frac{\partial E}{\partial \omega_{h12}}, \dots, \frac{\partial E}{\partial \omega_{hPH}}, \frac{\partial E}{\partial \omega_{o11}}, \frac{\partial E}{\partial \omega_{o12}}, \dots, \frac{\partial E}{\partial \omega_{oHL}}] \tag{2.17}$$

Here $\nabla E(\boldsymbol{\omega})$ is a vector composed of partial derivatives of E with respect to the components of vector $\boldsymbol{\omega}$ and specifies the steepest change direction of E in the error space. The partial derivatives are calculated as follows:

$$\begin{cases} \frac{\partial E}{\partial \omega_{hij}} = \delta_j x_i \ and \ \delta_j = o_j(1 - o_j) \sum_{k=1}^{L} \omega_{ojk} \delta_k \\ \frac{\partial E}{\partial \omega_{ojk}} = \delta_k o_j \ and \ \delta_k = o_k(1 - o_k)(y_k - o_k) \end{cases} \tag{2.18}$$

Here o_k is the output of unit $\{k\}_{k=1}^{L}$ in the output layer, and δ_k is the error term. o_j is the output of unit $\{j\}_{j=1}^{H}$ in the hidden layer, and δ_j is the error term. Given the direction of steepest change, the training rule of the weights for gradient descent is described by:

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \eta \nabla E(\boldsymbol{\omega}) \tag{2.19}$$

Here η is a positive constant called learning rate. The learning rate affects the speed of ANN reaching the minimum solution. Using Equation 2.19, each weight is updated as follows:

$$\begin{cases} \omega_{hij} \leftarrow \omega_{hij} + \eta \delta_j x_i \\ \omega_{ojk} \leftarrow \omega_{ojk} + \eta \delta_k o_j \end{cases} \tag{2.20}$$

In standard gradient descent, E is defined for all the training data D and the iteration through each training data in D has a fixed order. The weights in the network are updated at the end of iteration of D. While in stochastic gradient descent, E is defined for each training data d and the order of iterating through all training data in D is randomly shuffled. The weights are updated upon finishing training d. This results in a small weight update step comparing to the standard gradient descent. Stochastic gradient descent is more capable of avoiding the local minima during its search through the error space under the guidance of various $\{\nabla E_d(\boldsymbol{\omega})\}_{d \in D}$ rather than only a single $\nabla E_D(\boldsymbol{\omega})$.

The detailed BP algorithm in Figure 2.2 is as follows: we start the algorithm by initializing all network weights to small random numbers (e.g., between -0.05 and 0.05), learning rate to a positive small numbers (e.g., between 0.01 and 0.1), the maximum number of iteration and the error threshold. Then for each iteration of D, we first randomly shuffle the training data in D. Then for each training data d: $(\boldsymbol{x}, \boldsymbol{y})$, we input the $\boldsymbol{x}$ to the network and compute the output of every unit in the network using the sigmoid function. We then calculate the error term δ_k and δ_j using Equation 2.18. Given the error term of each unit, we update each weight using Equation 2.20. We then calculate the error of training data d as follows:

$$E_d = \sum_{k=1}^{L} \frac{1}{2} (y_{k_d} - o_{k_d})^2 \tag{2.21}$$

We continue to randomly shuffle all the training data, and update the weights through iteration of each training data. This process stops if a maximum number of iterations are reached or the error difference between adjacent iterations is below a given threshold.

2.3.3 Support Vector Machines (SVM)

SVM is used to construct a hyperplane or set of hyperplanes in a high- or infinite-dimensional space which can be used for classification (SVC) or regression (SVR). Consider a set of training data: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$, where $\boldsymbol{x}_i$ is a set of predictor variables $\{x_{j_i}\}_{j=1}^{P}$ and y_i is the target variable in training data $\{i\}_{i=1}^{N}$. In SVC, the target variable y_i is a class label and the hyperplane is a separation of different classes. The goal of SVC is to find a hyperplane that has the largest distance to the nearest training data point of any class. Support Vectors (SVs) are the nearest training data points of any class to the hyperplane, and margin is the perpendicular distance between the hyperplane and SVs. With a larger margin, the hyperplane has a higher probability to classify a new data point correctly. So the optimization problem of SVC is to maximize the margin subject to the constraints of correctly classifying all the training data. In SVR, the target variable y_i is a real value. The goal of SVR is to find a hyperplane (approximating function $\hat{f}$) that has at most ϵ deviation from true value y_i of the target variable for all the training data. For SVR, there exist two common approaches: ϵ-SVR and nu-SVR. In ϵ-SVR, ϵ is a pre-defined acceptable amount of deviation between $\hat{y}$ and y of the target variable. While in nu-SVR, ϵ is determined as a part of the learning algorithm. It has been demonstrated in [17] [18] that although ϵ-SVR is simpler, it has an equal or even better performance than nu-SVR. So in our work, we consider ϵ-SVR. In ϵ-SVR, the margin is defined as the perpendicular distance between hyperplane $\hat{f}$ and $\hat{f} \pm \epsilon$. Any data point in this margin is considered as well fitted by $\hat{f}$. So with a larger margin, the hyperplane $(\hat{f})$ has a higher probability to predict y of a new data point within an acceptable ϵ deviation. So the optimization problem of ϵ-SVR is to maximize the margin subject to the constraints of maximum ϵ deviation for all the training data.

We begin our description of ϵ-SVR by first applying to the case of linear approximating function $\hat{f}$. The approximating function $\hat{f}$ takes the form of:

$$\hat{f}(\boldsymbol{x}) = \langle \boldsymbol{\omega}, \boldsymbol{x} \rangle + b = \sum_{j=1}^{P} \omega_j x_j + b \tag{2.22}$$

Here ω_j is the weight of predictor variable x_j, and b is an intercept term. In this case, the margin is $\frac{\epsilon}{\|\omega\|^2}$, and $\|\boldsymbol{\omega}\|^2 = \langle \boldsymbol{\omega}, \boldsymbol{\omega} \rangle = \sum_{j=1}^{P} \omega_j \omega_j$. In the case of all the training data having at most ϵ deviation from $\hat{f}$, the optimization problem could be written as:

$$\min \frac{1}{2} \|\boldsymbol{\omega}\|^2 \tag{2.23}$$

$$\text{subject to} \begin{cases} y_i - \langle \boldsymbol{\omega}, \boldsymbol{x}_i \rangle - b \leq \epsilon \\ \langle \boldsymbol{\omega}, \boldsymbol{x}_i \rangle + b - y_i \leq \epsilon \\ \quad i = 1, \dots, N \end{cases}$$

Usually, not all the y of the training data can be well fitted using the approximation function $\hat{f}$ within an acceptable $\pm\epsilon$ deviation. In this case, we need to add other terms to the optimization problem in Equation 2.23 to allow for the training data with a deviation larger than ϵ. Slack variables [19] are introduced to represent the error for each training data: ξ_i as upper error bound and ξ_i^* as lower error bound. Then the error E_i of each training data $\{i\}_{i=1}^{N}$ is defined using the slack variables:

$$E_i = \begin{cases} \xi_i & y_i - \hat{f}(\boldsymbol{x}_i) > \epsilon \\ 0 & y_i - \hat{f}(\boldsymbol{x}_i) \in [-\epsilon, \epsilon] \\ \xi_i^* & y_i - \hat{f}(\boldsymbol{x}_i) < -\epsilon \end{cases} \tag{2.24}$$

Equation 2.23 is the modified to minimize the errors $\sum_{i=1}^{N} E_i$ as [20]:

$$\min \ \frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \tag{2.25}$$

$$\text{subject to} \begin{cases} y_i - \langle\boldsymbol{\omega}, \boldsymbol{x}_i\rangle - b \le \epsilon + \xi_i \\ \langle\boldsymbol{\omega}, \boldsymbol{x}_i\rangle + b - y_i \le \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0, i = 1, \dots, N \end{cases}$$

Here C is pre-defined value adjusting the trade-off between the margin of $\hat{f}$ and the tolerance of deviation larger than ϵ. To solve the optimization problem given in Equation 2.25 with corresponding constraints, the Lagrange method with Lagrange multipliers is used. The Lagrange function is described as follows:

$$L \coloneqq \frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) - \sum_{i=1}^{N}\alpha_i(\epsilon + \xi_i^* - y_i + \langle\boldsymbol{\omega}, \boldsymbol{x}_i\rangle + b) - \sum_{i=1}^{N}\alpha_i^*(\epsilon + \xi_i^* + y_i - \langle\boldsymbol{\omega}, \boldsymbol{x}_i\rangle - b) - \sum_{i=1}^{N}(\eta_i\xi_i + \eta_i^*\xi_i^*) \tag{2.26}$$

$$\text{subject to } \eta_i, \eta_i^*, \alpha_i, \alpha_i^* \ge 0$$

Here L is Lagrangian, and $\eta_i, \eta_i^*, \alpha_i, \alpha_i^*$ are Lagrange multipliers. For the optimality of Equation 2.26, the partial derivatives of L with respect to the variables $(\omega, b, \xi_i, \xi_i^*)$ should be set to zero, which are given as follows:

$$\begin{cases} \partial_b L = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \\ \partial_{\boldsymbol{\omega}} L = \boldsymbol{\omega} - \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\boldsymbol{x}_i = 0 \\ \partial_{\xi_i} L = C - \alpha_i - \eta_i = 0 \\ \partial_{\xi_i^*} L = C - \alpha_i^* - \eta_i^* = 0 \end{cases} \tag{2.27}$$

ω_j is a linear combination of x_j of all the training data. $\boldsymbol{\omega}$ is given by

$$\boldsymbol{\omega} = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\boldsymbol{x}_i \tag{2.28}$$

Then $\hat{f}$ becomes:

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\langle \boldsymbol{x}_i, \boldsymbol{x}\rangle + b = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\sum_{j=1}^{P} x_{j_i} x_j + b \tag{2.29}$$

Substituting Equation 2.27 into Equation 2.26, the optimization problem in Equation 2.25 is then transformed to its dual optimization problem:

$$\max\ -\frac{1}{2}\sum_{i,k=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_k - \alpha_k^*)\langle \boldsymbol{x}_i, \boldsymbol{x}_k\rangle - \epsilon\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*) \tag{2.30}$$

$$\text{subject to} \begin{cases} \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \\ \alpha_i, \alpha_i^* \in [0, C], i = 1, \dots, N \end{cases}$$

Here $\langle \boldsymbol{x}_i, \boldsymbol{x}_k\rangle = \sum_{j=1}^{P} x_{j_i} x_{j_k}$. Then values of $\{\alpha_i\}_{i=1}^{N}$ or $\{\alpha_i^*\}_{i=1}^{N}$ are obtained by maximizing the objective function subject to corresponding constraints in the dual optimization problem. To solve this dual optimization problem, a Sequential Minimal Optimization (SMO) method is normally used, which decomposes the overall problem into sub-problems for computational efficiency. At each iteration, SMO selects two Lagrange multiplier α at a time while keeping other α fixed. Then these two multipliers are optimized using the objective function in the dual problem. The objective function is then updated with the new optimal α. This process is repeated until the optimization of the objective function converges.

In computing b, Karush-Kuhn-Tucker(KKT) [21] optimality conditions are used. The KKT conditions of our problem state that at the optimal solution, the product between the Lagrange multipliers and constraints should be set to zero. In Equation 2.26 this means:

$$\begin{cases} \alpha_i(\epsilon + \xi_i^* - y_i + \langle \boldsymbol{\omega}, \boldsymbol{x}_i \rangle + b) = 0 \\ \alpha_i^*(\epsilon + \xi_i^* + y_i - \langle \boldsymbol{\omega}, \boldsymbol{x}_i \rangle - b) = 0 \end{cases} \tag{2.31}$$

$$\begin{cases} \eta_i \xi_i = (C - \alpha_i)\xi_i = 0 \\ \eta_i^* \xi_i^* = (C - \alpha_i^*)\xi_i^* = 0 \end{cases} \tag{2.32}$$

Equation 2.31 shows that α_i and α_i^* cannot both simultaneously be non-zero, and only training data i with $\left|y_i - \hat{f}(\boldsymbol{x}_i)\right| \geq \epsilon$ may have a non-zero α_i or α_i^*. Therefore, in Equation 2.28, $\boldsymbol{\omega}$ is determined only by the training data with non-zero Lagrangian multiplier α_i or α_i^*, and these training data are called Support Vectors (SVs) in SVR. Equation 2.32 shows that only training data $(\boldsymbol{x}_i, y_i)$ with corresponding $\alpha_i = C$ or $\alpha_i^* = C$ may have a non-zero ξ_i or ξ_i^*, which means that these training data lie outside the margin. So to determine b, we can only use the training data with exactly ϵ deviation from $\hat{f}$, and these training data have $\alpha_i \in (0, C)$ or $\alpha_i^* \in (0, C)$. Hence, b is determined as follows:

$$\begin{cases} b_i = y_i - \langle \boldsymbol{\omega}, \boldsymbol{x}_i \rangle - \epsilon & \text{for } \alpha_i \in (0, C) \\ b_i = y_i - \langle \boldsymbol{\omega}, \boldsymbol{x}_i \rangle + \epsilon & \text{for } \alpha_i^* \in (0, C) \end{cases} \tag{2.33}$$

For numerical stability, the b value is an average of all the b_i values.

The SVR above is only applicable to a linear relationship of $(\boldsymbol{x}_i, y_i)$ among the training data. However, SVR can be extended to model non-linear training data utilizing kernel trick. In machine learning, kernel trick [22] is a method to map the observations (training data) in data space $\boldsymbol{X}$ to an inner product space $\boldsymbol{H}$ of a higher dimension, where the observations have a linear relationship. Here the dimension of $\boldsymbol{X}$ is the number of predictor variables P, and the dimension of $\boldsymbol{H}$ is the number of training data N. The kernel trick is denoted as:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_k) \equiv \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_k) \rangle \tag{2.34}$$

Here Φ is a mapping function: $\boldsymbol{X} \rightarrow \boldsymbol{H}$. The kernel trick helps in calculating the dot product in high dimension space without having to explicitly compute Φ . Different kernel functions could be used for different problems, such as Polynomial kernel, Gaussian kernel (also called Radial Basis Function kernel) or Sigmoid kernel [23]. For example, in Gaussian kernel, $k(\boldsymbol{x}_i, \boldsymbol{x}_k)$ is calculated as follows:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_k) = Gaussian(\boldsymbol{x}_i, \boldsymbol{x}_k) = \exp\{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2\} \tag{2.35}$$

Here exp is the exponential function and γ is a pre-defined value which determines the Gaussian kernel width. $\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2$ is the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_k$. This distance is normally captured using Euclidean distance:

$$\|\boldsymbol{x}_i - \boldsymbol{x}_k\|^2 = \sqrt{\textstyle\sum_{j=1}^{P} \left(x_{j_i} - x_{j_k}\right)^2} \tag{2.36}$$

Thus for non-linear modeling, the SVR minimization equation in Equation 2.30 can be rewritten as follows:

$$\text{maximize} -\frac{1}{2} \sum_{i,k=1}^{N} (\alpha_i - \alpha_i^*)(\alpha_k - \alpha_k^*) k(\boldsymbol{x}_i, \boldsymbol{x}_k) - \epsilon \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) + \sum_{i=1}^{N} y_i (\alpha_i - \alpha_i^*)$$

$$\text{subject to} \begin{cases} \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) = 0 \\ \alpha_i, \alpha_i^* \in [0, C], i = 1, \dots, N \end{cases} \tag{2.37}$$

The hyperplane $\hat{f}$ is then given by**:**

$$\hat{f}(\boldsymbol{x}) = \textstyle\sum_{i=1}^{N} (\alpha_i - \alpha_i^*) k(\boldsymbol{x}_i, \boldsymbol{x}_k) + b \tag{2.38}$$

2.3.4 Kernel Canonical Correlation Analysis (KCCA)

Consider a set of N training data: $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \dots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$, where $\boldsymbol{x}_i$ is a set of predictor variables $\{x_{j_i}\}_{j=1}^{P}$ and $\boldsymbol{y}_i$ is a set of target variables $\{y_{l_i}\}_{l=1}^{L}$. Canonical Correlation Analysis (CCA) seeks to find a $P \times 1$ column vector $\boldsymbol{\omega_x}$ and a $L \times 1$ column vector $\boldsymbol{\omega_y}$, such that the correlation ρ between the projection of $\{\boldsymbol{x}_i\}_{i=1}^{N}$ on $\boldsymbol{\omega_x}$ and $\{\boldsymbol{y}_i\}_{i=1}^{N}$ on $\boldsymbol{\omega_y}$ are mutually maximized. ρ is defined as follows:

$$\rho\left(\boldsymbol{\omega}_x^{\boldsymbol{T}}\boldsymbol{x}, \boldsymbol{\omega}_y^{\boldsymbol{T}}\boldsymbol{y}\right) = \frac{\operatorname{cov}(\boldsymbol{\omega}_x^{\boldsymbol{T}}\boldsymbol{x}, \boldsymbol{\omega}_y^{\boldsymbol{T}}\boldsymbol{y})}{\sqrt{\operatorname{var}\left(\boldsymbol{\omega}_x^{\boldsymbol{T}}\boldsymbol{x}\right)\operatorname{var}\left(\boldsymbol{\omega}_y^{\boldsymbol{T}}\boldsymbol{y}\right)}} = \frac{\boldsymbol{\omega}_x^{\boldsymbol{T}} C_{xy}\boldsymbol{\omega}_y}{\sqrt{(\boldsymbol{\omega}_x^{\boldsymbol{T}} C_{xx}\boldsymbol{\omega}_x)(\boldsymbol{\omega}_y^{\boldsymbol{T}} C_{yy}\boldsymbol{\omega}_y)}} \tag{2.39}$$

Here $\boldsymbol{\omega}_x^{\boldsymbol{T}}$ is the transpose of $\boldsymbol{\omega_x}$ and $\boldsymbol{\omega}_y^{\boldsymbol{T}}$ is the transpose of $\boldsymbol{\omega_y}$. $C_{\boldsymbol{xy}}$ is the covariance $\operatorname{cov}(\boldsymbol{x}, \boldsymbol{y})$. $C_{\boldsymbol{xx}}$ and $C_{\boldsymbol{yy}}$ are the covariance $\operatorname{cov}(\boldsymbol{x}, \boldsymbol{x})$ and $\operatorname{cov}(\boldsymbol{y}, \boldsymbol{y})$ respectively. The optimization problem of CCA is defined as follows:

$$\max_{\boldsymbol{\omega}_x, \boldsymbol{\omega}_y} \quad \rho(\boldsymbol{\omega}_x^{\boldsymbol{T}}\boldsymbol{x}, \boldsymbol{\omega}_y^{\boldsymbol{T}}\boldsymbol{y}) \tag{2.40}$$

$$\text{subject to } \begin{cases} \operatorname{var}(\boldsymbol{\omega}_x^{\boldsymbol{T}}\boldsymbol{x}) = 1 \\ \operatorname{var}\left(\boldsymbol{\omega}_y^{\boldsymbol{T}}\boldsymbol{y}\right) = 1 \end{cases}$$

This optimization problem can be solved by Lagrange method utilizing the Lagrange multipliers [24]. The optimization problem in Equation 2.40 is then reduced to the following generalized eigenvalue problem:

$$\begin{pmatrix} \boldsymbol{0} & C_{xy} \\ C_{yx} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \end{pmatrix} = \rho \begin{pmatrix} C_{xx} & \boldsymbol{0} \\ \boldsymbol{0} & C_{yy} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \end{pmatrix} \tag{2.41}$$

This optimization problem has $P + L$ eigenvalues and the corresponding eigenvector of each eigenvalue is a $(P + L) \times 1$ column vector. $\boldsymbol{\omega_x}$ and $\boldsymbol{\omega_y}$ correspond to the top P elements and bottom L elements of the eigenvector of the largest eigenvalue respectively. The projection of $\boldsymbol{x}$ on $\boldsymbol{\omega_x}$ and the projection of $\boldsymbol{y}$ on $\boldsymbol{\omega_y}$ are mutually maximally correlated.

CCA is only useful if there exists a linear relationship between predictor variables $\boldsymbol{x}$ and target variables $\boldsymbol{y}$. However, CCA can be extended to Kernel CCA (KCCA), which models non-linear training data utilizing a kernel trick similar to that used in Section 2.3.3. The training data is mapped to a higher dimensional space where a linear relationship exists between the predictor variables and target variables. In Equation 2.34 a Gaussian kernel is used to generate a $N \times N$ predictor similarity matrix $\boldsymbol{K_x}$ whose $(i,k)^{th}$ entry measures the similarity between $\boldsymbol{x}_i$ and $\boldsymbol{x}_k$, and a $N \times N$ target similarity matrix $\boldsymbol{K_y}$ whose $(i,k)^{th}$ entry is the similarity between $\boldsymbol{y}_i$ and $\boldsymbol{y}_k$. This process is described as follows:

$$\begin{cases} \boldsymbol{K_x}[i,k] = k(\boldsymbol{x}_i, \boldsymbol{x}_k) = Gaussian(\boldsymbol{x}_i, \boldsymbol{x}_k) = \exp\{-\gamma \lVert \boldsymbol{x}_i - \boldsymbol{x}_k \rVert^2\} \\ \boldsymbol{K_y}[i,k] = k(\boldsymbol{y}_i, \boldsymbol{y}_k) = Gaussian(\boldsymbol{y}_i, \boldsymbol{y}_k) = \exp\{-\gamma \lVert \boldsymbol{y}_i - \boldsymbol{y}_k \rVert^2\} \end{cases} \tag{2.42}$$

Here exp is the exponential function and γ is a pre-defined value which determines the Gaussian kernel width. $\lVert \boldsymbol{x}_i - \boldsymbol{x}_k \rVert^2$ is the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_k$, and $\lVert \boldsymbol{y}_i - \boldsymbol{y}_k \rVert^2$ is the distance between $\boldsymbol{y}_i$ and $\boldsymbol{y}_k$. These distances are normally computed using Euclidean distance:

$$\begin{cases} \lVert \boldsymbol{x}_i - \boldsymbol{x}_k \rVert^2 = \sqrt{\sum_{j=1}^{P} \left(x_{j_i} - x_{j_k}\right)^2} \\ \lVert \boldsymbol{y}_i - \boldsymbol{y}_k \rVert^2 = \sqrt{\sum_{l=1}^{L} \left(y_{l_i} - y_{l_k}\right)^2} \end{cases} \tag{2.43}$$

For the two kernelized matrices $K_{\boldsymbol{x}}$ and $K_{\boldsymbol{y}}$, the optimization problem of CCA in Equation 2.40 is extended to [25]:

$$\max_{\boldsymbol{\alpha_x}, \boldsymbol{\alpha_y}} \quad \rho\left(\boldsymbol{\alpha_x^T K_x}, \boldsymbol{\alpha_y^T K_y}\right) = \frac{\boldsymbol{\alpha_x^T K_x K_y \alpha_y}}{\sqrt{\mathrm{var}\left(\boldsymbol{\alpha_x^T K_x}\right)\mathrm{var}\left(\boldsymbol{\alpha_y^T K_y}\right)}} \tag{2.44}$$

Here $\boldsymbol{\alpha_x}$ is a $N \times N$ matrix and $\boldsymbol{\alpha_x^T}$ is the transpose of $\boldsymbol{\alpha_x}$. $\boldsymbol{\alpha_y}$ is a $N \times N$ matrix and $\boldsymbol{\alpha_y^T}$ is the transpose of $\boldsymbol{\alpha_y}$. The optimization problem for KCCA is:

$$\begin{pmatrix} \mathbf{0} & \boldsymbol{K_x K_y} \\ \boldsymbol{K_y K_x} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \alpha_x \\ \alpha_y \end{pmatrix} = \rho \begin{pmatrix} \boldsymbol{K_x K_x} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{K_y K_y} \end{pmatrix} \begin{pmatrix} \alpha_x \\ \alpha_y \end{pmatrix} \tag{2.45}$$

This optimization problem has $2N$ eigenvalues and the corresponding eigenvector of each eigenvalue is a $2N \times 1$ column vector. $\boldsymbol{\alpha_x}$ and $\boldsymbol{\alpha_y}$ correspond to the top N rows and bottom N rows of the eigenvectors of the largest N eigenvalues respectively. The projection of $\boldsymbol{K_x}$ on $\boldsymbol{\alpha_x}$ and the projection of $\boldsymbol{K_y}$ on $\boldsymbol{\alpha_y}$ are mutually maximally correlated.

For a test data point, the associated values of the target variables are estimated by applying the kernel function and distance metric of Equation 2.42 and Equation 2.43 to generate a $1 \times N$ row vector $\boldsymbol{v}$. This vector captures the similarities of the predictor variables between the test data point and all the training data $\{\boldsymbol{x}_i\}_{i=1}^{N}$. The vector $\boldsymbol{v}$ is then projected on $\boldsymbol{\alpha_x}$ and its k nearest neighbours in $\boldsymbol{\alpha_x^T K_x}$ are determined. The values of target variables are then averaged to generate the prediction of the target variables for the test data point.

2.4 Evaluation Methodology

We present an evaluation framework for quantitatively comparing the performance of statistical machine learning algorithms in modeling chip multiprocessor architectures. The machine learning model seeks to capture the dependence of one or more performance metrics on multiple architectural parameters of interest. The flow chart in Figure 2.4 outlines the different components of our proposed framework.

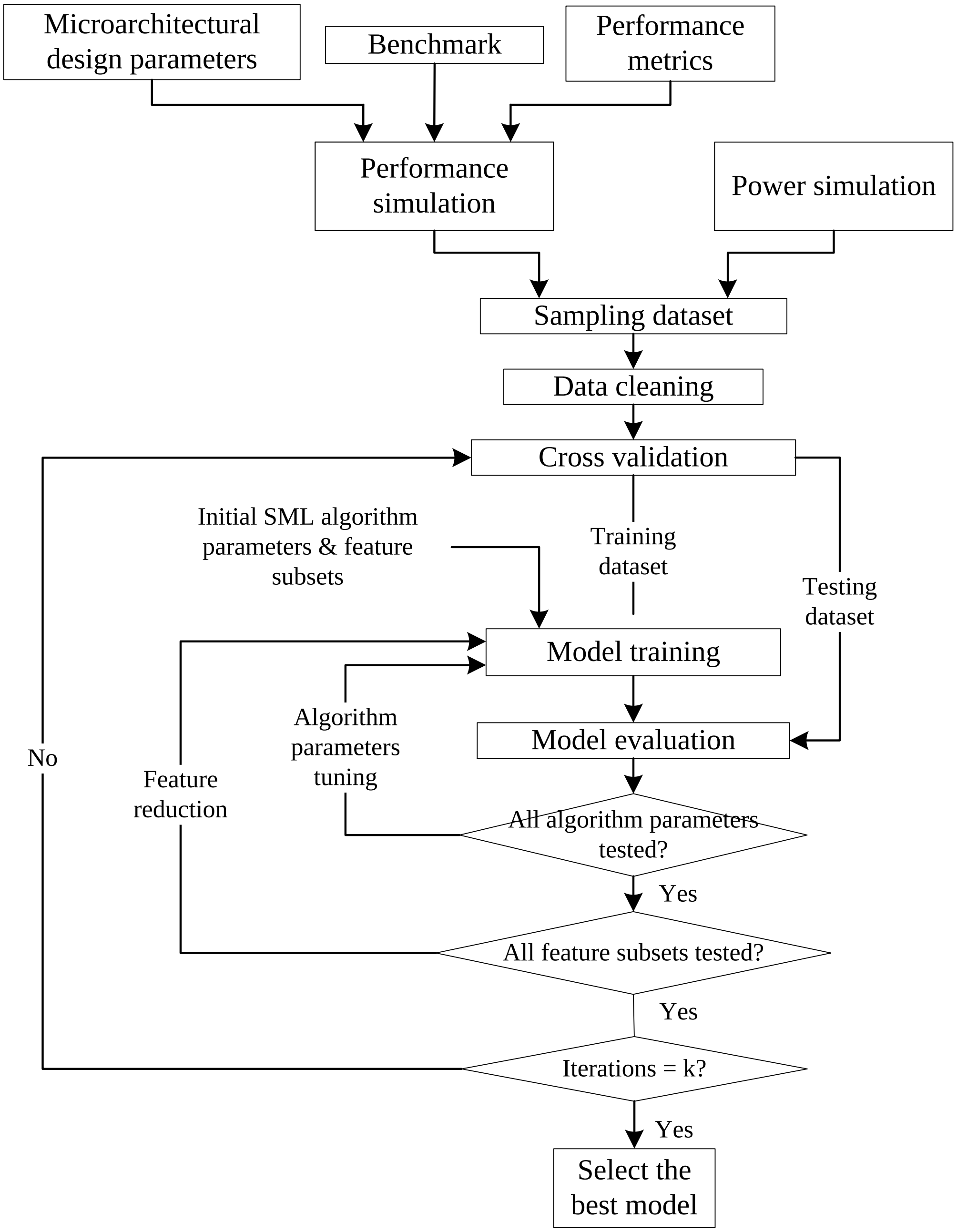

Figure 2.4: Flowchart for multi-core power and performance SML model generation

We first identify the architectural parameters that serve as the model predictor variables, and the performance metrics as the model outputs. We then choose a benchmark of interest and an architecture simulation tool capable of executing the benchmark to the desired details for different values of the predictor variable. To construct a dataset for modeling, we use data sampling techniques to choose a subset configuration of the feasible values of input parameters, and then simulate these to obtain the associated performance metrics. The sampled dataset is cleaned to remove outliers and missing values. We then identify the significance of each predictor variable and select a subset of most relevant architectural parameters to build the learning models. Subsequently, the performance of the model is evaluated for a test dataset. In the following section, the different components of the evaluation framework described above are presented in detail.

2.4.1 Target Architecture and Design Space

The SML algorithm evaluation framework of Figure 2.4 can be applied to any architecture of interest. However, to keep our discussions concrete, in our work we choose a multi-core architecture common in today’s server class processors. We seek to model a quad-core processor, with each core having a private L1 cache, and all cores sharing the L2 cache through a shared bus. The individual cores are out-of-order and based on the Alpha 21264 processor. Each out-of-order core is a single threaded four issue processor with split instruction and data caches. The L1 and L2 caches are non-blocking with miss status holding register (MSHR) and write buffers (WB) for read and write misses. Cache coherency is maintained among the 4 cores sharing the L2 cache. The cache replacement policy is LRU and the cache coherence protocol is bus-based MOESI snooping protocol. Our local memory system is a classic bus based model. The bus arbitration follows first-come-first-serve logic, and uses round-robin scheduling for bus accesses.

The architectural design parameters of interest are grouped as core and memory. Core parameters include voltage/frequency, the number of integer and floating point units, integer and floating point register sizes, instruction and store queue sizes; the memory parameters include L1 and L2 cache size and associativity, and DTLB an ITLB sizes. The complete list of parameters chosen in our work is given in Table 2.1. The possible range of values assumed by the core, memory and network parameters are derived from for commercial processors and from the literature. We choose power and benchmark specific performance metrics such as task throughput as the output performance metrics. Note that our choice of the architectural parameters, their possible range of values, and the output performance metrics are only illustrative. Typically, designers would use application specific performance goals and knowledge of the workload to choose these metrics.

Table 2.1: List of micro-architectural model predictors

Parameters	**Value ranges**
Number of integer Alu-MultDiv	2-1, 4-1, 4-2
Number of floating point Alu-MultDiv	1-1, 2-1, 4-2
Number of local-global history bits	2**(11-12, 11-13, 12-13)
Number of physical integer registers	64, 128, 196
Number of physical floating point registers	64, 128, 196
Number of store-load queue entries	32-48, 32-64, 48-64
Number of instruction queue entries	16, 32
Number of reorder buffer entries	128, 160
Number of branch target buffer Entries	2048, 4096
Return address stack size	16, 32
L1 cache size (KB)	32, 64
L1 instruction cache associativity	4, 8
L1 data cache associativity	4, 8
DTLB size	32, 64, 128
ITLB size	32, 64, 128
Core frequency (GHz)	0.8, 1.0, 1.33, 1.8, 2.0, 2.33, 2.66
L2 cache size (MB)	1, 2, 4
L2 cache associativity	8, 16
Back side bus frequency	Core frequency
Front side bus frequency (MHz)	100, 133
Bus Width (Byte)	8, 16, 24

2.4.2 Benchmarks

We use multimedia, business, and data mining benchmarks obtained from the Parsec suite. The Princeton Application Repository for Shared-Memory Computers (PARSEC) [3] is a benchmark suite composed of multithreaded programs. The suite focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for chip-multiprocessors. A summary of the benchmarks used in our work is given in Table 2.2. The benchmarks are cross-compiled using cross-tool-NG 1.5.2 for the Alpha architecture to run on Gem5.

Table 2.2: Summary of the simulated benchmarks with descriptions

Domain	**Benchmark**	**Description**
Media processing	x264	An H.264/AVC video encoding application
Rendering	Raytrace	An animated real-time 3D scene rendering application that uses ray tracing method
Financial Analysis	Blackscholes	Option pricing kernel that uses the Black-Scholes partial differential equation
	Swaptions	Application which prices a portfolio of swaptions with the Heath-Jarro-Morton framework
Data Mining	Freqmine	A frequent itemset mining application which uses an array-based version of the Frequent Pattern-growth method
	Streamcluster	Online streaming input points clustering kernel

2.4.3 Processor Simulation Tools

2.4.3.1 Processor Performance Modeling

We use the Gem5 simulator [26] for the performance modeling of our target system. Gem5 is an event-driven cycle-accurate simulator, which provides a highly configurable simulation framework, multiple ISAs, diverse CPU models, multiple cache coherence protocols and interconnects models. Gem5 provides a simple, functional, one-

CPI CPU and a detailed model of an out-of-order SMT-capable CPU. It also supports both full-system and system-call emulation modes. The full-system simulation mode simulates a complete computer system including operating system kernel and I/O devices, while in system-call emulation mode, the common system calls are emulated by the host OS. Gem5 provides a clear interface for check-pointing, fast-forwarding, debugging and statistics. We model our target system with the Alpha ISA (instruction set architecture) as Alpha ISA is the most well developed and stable ISA supported in Gem5. Our many-core processor operates in full-system mode, which models a complete Linux system with kernel version 2.6.27. We run the selected benchmarks in Gem5 with problem size of each benchmark shown in Table 2.3. We fast forward the simulation starting point to the pre-defined start of region of interest (ROI), where the parallel execution phases of each benchmark commences. We simulate the whole ROI for x264 and Raytrace separately, while Blackscholes/swaprtions and Freqmine/streamcluster are run together on the 4 cores. The performance metrics for each benchmark are shown in Table2.3. Instead of the commonly used IPC or CPI we use more application oriented performance metrics such as FPS (frame per second) and runtime in seconds. The use of application oriented

Table 2.3: Benchmark problem size, performance metric, and simulation time for detailed simulation

Benchmark	**Problem size**	**Performance metric**	**Simulation time (hours)**
x264	360x240 pixels, 8 frames	FPS	3
Raytrace	480x270 pixels, 3 frames, 1 million polygon	FPS	7
Blackscholes & Swaptions	4,096 options & 16 swaptions, 5000 simulations	Runtime	4
Freqmine & streamcluster	10,000 transactions & 2,048 points per block, 1 block	Runtime	6.5

performance metrics is especially important for multithreaded and real-time workloads. [27].

2.4.3.2 Processor Power Modeling

We use McPAT 0.8 [28] for the power modeling of our target system. McPAT is an integrated power, area, and timing hierarchical modeling framework for multi-core and many-core processor configurations from 90nm to 22nm. McPAT models the power at the micro-architectural level, circuit level and technology level.

At the micro-architectural level, McPAT includes models of major architectural components such as cores, interconnection networks, caches, memory controllers, and clocking. At the circuit level, the architectural building blocks are mapped into four basic circuit structures: hierarchical wires, arrays, complex logic, and clocking networks. At the technology level, the physical parameters of devices and wires, such as unit resistance, capacitance, and current density, is calculated based on the data from the ITRS roadmap.

We model our cores and L2 caches using the 22nm technology node across all the benchmarks. The system configurations and performance statistics from Gem5 are input into McPAT through an XML interface to compute the static and dynamic power dissipation of the cores and L2 caches. The total power dissipation is the sum of runtime-dynamic power dissipation and leakage power dissipation of the cores and L2 cache.

2.4.4 Data Sampling

Consider the design parameters (predictor variables) in the chip multiprocessor design as shown in Table 2.1 with a range of possible values. The combinations of the possible values of all the design parameters constitute the overall design space. A micro-architectural cycle-accurate simulation is required to obtain the values of the performance metrics and power (target variables) of each design point. Since each design point requires several hours of simulation time, it is extremely expensive and even infeasible to simulate all the design points especially for a design space with billions of points. In a modeling based approach a predictive model is generated from a training subset generated by sampling the overall design space. The quality of the selected design points is very critical, because all the inferences and predictions of the test design points are based on them. The commonly used sampling methods include random sampling, systematic sampling [29] and stratified sampling [30]. In random sampling the design points are randomly generated. In systematic sampling, the selection of the design points is based on a fixed distance metric between the design points. In stratified sampling the design points are divided into subgroups and the selection of design points from each subgroup is done by either random sampling or systematic sampling. However, none of these methods guarantee an even distributed sampling across the overall design space.

Recently, Latin Hypercube Sampling (LHS) has been proposed by researchers [31] [32] as an alternative sampling method. In LHS, to obtain N design points from a P-dimensional design space, each dimension is divided by N hyperplanes resulting in a total number of N^P small hypercubes. At most one design point is selected from each hypercube. The design point is selected by either Maximum Minimum (*MaxiMin*) Distance LHS criteria or Minimum Correlation (*Correlation*) LHS criteria. In *MaxiMin* the minimum distance between the selected design points is maximized. The distance can be calculated using a distance metric, such as the Euclidean distance. In *Correlation* the correlation between the selected design points is minimized. The correlation can be calculated using the canonical correlation calculation as described in Equation 2.39. These two criteria enable LHS to guarantee a relatively uniformly distributed design points to be sampled from the design space. In our work, we choose the *MaxiMin* Distance LHS criteria implemented by a Matlab function: *lhsdesign* [33]. The output of *lhsdesign* is a normalized $N \times P$ matrix. We denormalize the entries in this matrix to the possible values in the range of each predictor variable shown in Table 2.1.

In our work, we divide the micro-architectural parameters into processing core level parameters and memory-interconnect level parameters. We sample points from the design space of each level respectively. We select 832 (~0.5% of total) points from the design space of core-level parameters by using *MaxiMin* with memory-interconnect parameters fixed. The fixed memory-interconnect parameters are shown in Table 2.5. We select all (72) the points from the design space of memory-interconnect parameters with core-level parameters fixed. The fixed core-level parameters are shown in Table 2.6. To capture the interdependences between the core and memory-interconnect parameters, we sample an additional 150 (0.00025%) points from the total design space using *MaxiMin*. The training dataset is then obtained through gem5/McPAT micro-architectural simulation of the selected design configurations. Our choice of the number of design points is guided by the practical requirement to keep the simulation time reasonable.

Table 2.4: Fixed interconnect-memory level micro-architectural design parameters

Parameters	**Value ranges**
Core frequency (GHz)	1.8
L2 cache size (MB)	2
L2 cache associativity	8
Back side bus frequency	Core frequency
Front side bus frequency (MHz)	133
Bus Width (Byte)	8

Table 2.5: Fixed CMP core-level micro-architectural design parameters

Parameters	**Value ranges**
Number of integer Alu-MultDiv	4-1
Number of floating point Alu-MultDiv	2-1
Number of local-global history bits	2**(11-13)
Number of physical integer registers	128
Number of physical floating point registers	64
Number of store-load queue entries	32-64
Number of instruction queue entries	16
Number of reorder buffer entries	160
Number of branch target buffer Entries	2048
Return address stack size	32
L1 cache size (KB)	32
Instruction cache associativity	4
Data cache associativity	8
ITLB size	32
DTLB size	64

2.4.5 Data Cleaning

The raw samples may have missing values and outliers caused due to invalid combination of the values of the predictor variables or simulation artifacts. Data cleaning eliminates or fills in missing values and removes outliers from the raw samples resulting in a more accurate model. Missing values are commonly filled by using a predefined global constant, the target variable's mean value, or a most probable value obtained by observing the values of the target variables of other samples [34]. In our work, we fill in the missing value by rerunning the simulations or by eliminating the invalid combination of the input values.

The outliers are commonly identified by either distance-based methods [35] or clustering-based methods [36]. In distance-based methods, the set of k nearest neighbouring samples of each sample is obtained by using a distance metric. For this neighboring set, the outlier is defined as the sample that lies greater than a predefined distance. In clustering-based methods, the samples are partitioned into clusters, and the outlier is defined as the sample that does not fit well with any cluster. In our work, we assume that a similar combination of values of micro-architectural design parameters would have a similar combination of values of performance metrics. We then choose a clustering-based method to identify outliers, and use the K-Means [37] clustering algorithm. Consider N training data $\{\boldsymbol{X}, \boldsymbol{Y}\}: \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \dots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$, where $\boldsymbol{x}_i$ is a set of predictor variables $\{x_{j_i}\}_{j=1}^{P}$ and $\boldsymbol{y}_i$ is a set of target variables $\{y_{l_i}\}_{l=1}^{L}$. The goal of the K-Means algorithm is to partition the N training data into K clusters, such that the total within-cluster distance given by Equation 2.46 is minimized:

$$\min \sum_{k=1}^{K} \sum_{\boldsymbol{y}_i \in S_k} ||\boldsymbol{y}_i - \boldsymbol{u}_k||^2 \tag{2.46}$$

Here S_k is the set of training data in cluster k and $\boldsymbol{u}_k$ is the mean vector (centroid). The distance $||\boldsymbol{y}_i - \boldsymbol{u}_k||^2$ is calculated by using a distance metric such as the Euclidean distance. To determine the centroids $\{\boldsymbol{u}_k\}_{k=1}^{K}$, K-Means algorithm first initializes K centroids which are randomly selected from the training data. Then for each iteration, $\{\boldsymbol{y}_i\}_{i=1}^{N}$ is assigned to the cluster k, such that the distance to the centroid $\boldsymbol{u}_k$ is minimized. After each $\boldsymbol{y}_i$ has been assigned to different clusters, the centroids of all the clusters are updated. The assignment process is repeated until all the centroids are stable. In our work, we choose the *kmeans* library in R [38] which is an implementation of the above K-Means algorithm. To determine the number of clusters K for a set of training data, we evaluate the total within-cluster distances for a range of $\{K\}_{K=2}^{40}$ using the K-Means algorithm. Then we choose K such that $K + 1$ does not significantly decrease the total within-cluster distances.

Given the number of clusters K and the values of the target variables in all the training data $\boldsymbol{Y}: \{\boldsymbol{y}_i\}_{i=1}^{N}$, the outliers are determined using K-Means algorithm iteratively. For each iteration, the centroid of each cluster is calculated by applying the K-Means to $\boldsymbol{Y}$. Then the maximum distance d_{max} among all the distances between $\boldsymbol{y}_i \in \boldsymbol{Y}$ and its centroid is obtained. The $\boldsymbol{y}_i \in \boldsymbol{Y}$ that has a distance to its centroid larger than $T \times d_{max}$ is considered an outlier in this iteration, and removed from $\boldsymbol{Y}$. Here T is a predefined distance threshold and $T < 1$. This process continues until a predefined number of iterations (I) is reached. In our work, we find that $T = 0.95$ and $I = 30$ are reasonable values based on the distribution of the samples in each cluster. By applying the above described outlier identification methods, we eliminate about 5% of the training data set.

2.4.6 Feature Reduction and SML Algorithms Parameters Tuning

Training a learner with all the possible predictor variables (features) does not necessarily produce the best learner, as the weakly relevant, irrelevant and redundant predictor variables can negatively influence the effectiveness and performance of the SML algorithms. Feature reduction is a procedure to identify the significance of each predictor variable, and select a subset of most relevant predictor variables to build the learning models for given target variables. Consider training data with P predictor variables. To select the optimal subset of predictor variables, a total number of $P!$ possible subsets need to be evaluated. Such an exhaustive search is only feasible for a small P. Heuristic methods that search through a reduced number of subsets are commonly used for variable selection. Examples of heuristic methods for variable selection are stepwise forward selection/backward elimination methods [39], genetic algorithms [40] and simulated annealing [41].

The stepwise forward selection/backward elimination methods are the most commonly used methods [42]. In stepwise forward selection, the procedure starts with an empty set of predictor variables as the current subset. At each iteration, the model with current subset of predictor variables is first evaluated using the model independent metric. Such a metric could be the t-statistic value for linear models or R^2 values. Then each unselected predictor variable is added to the current subset separately and the model is re-evaluated using the same metric. The unselected predictor variable which leads to the largest change of the metric is considered the most significant predictor variable in this iteration and added to current subset for next iteration. In stepwise backward elimination, the procedure starts with the full set of predictor variables as the current subset. At each iteration, the model with the current subset of predictor variables is first evaluated using the model independent metric as mentioned above. Then each predictor variable is eliminated from the current subset separately and the model is re-evaluated using the same metric. The predictor variable which leads to the smallest change of the metric is considered the least significant predictor variable in this iteration and eliminated from current subset for next iteration. For these two methods, the selection process or the elimination process stops when a threshold on the model accuracy has been reached, or all the predictor variables have been added to or eliminated from current subset. The optimal subset of predictor variables will be the one with the most accurate model. Therefore, for these two methods, if all the predictor variables are considered during the stepwise process, the total number of possible subsets need to be evaluated is $\frac{1}{2}P^2$.

In our work, we choose the *rfe* function provided in the *caret* R package [43] which uses the stepwise backward elimination with k-fold cross validation. The k-fold cross validation provides a better estimation of the performance of models. In k-fold cross validation, the training data is first partitioned into equal-sized k folds. The model training process is then performed in successive k rounds. In each round, a different fold is selected for model evaluation using the metric mentioned above and the other $k-1$ folds are used for model training. This results in k lists of significance of every predictor variable in each iteration of the elimination process. The least significant predictor variable is selected based on an average of the significance in the k lists. The k round can be also performed in parallel to alleviate the computational burden. Another advantage of the implementation of variable selection in the *caret* package is that it can combine the parameter tuning of each machine learning algorithm along with the seeking of optimal subset of predictor variables. The parameters that can be tuned for each machine learning algorithm is shown in Table 2.6 with corresponding possible values. In the elimination process of this package, all the predictor variables are removed from the full set iteratively to evaluate the learning model accuracy with all the possible values of the tunable parameters. The metric is RMSE (root mean squared error) and R^2 value of the selected target variable, which are described as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}_i)^2} \tag{2.47}$$

$$R^2 = \frac{\sum_{i=1}^{N}(y_i - \hat{f}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{2.48}$$

Here N is the number of training data, y_i is the actual value of the selected target variable in training data i, $\hat{f}_i$ is the estimated value of the selected target variable, and $\bar{y}$ is the mean of actual values of the selected target variable in all the training data.

It should be noted that for each tunable parameter, the possible values listed in Table 2.6 are only a fraction of all the possible values for that parameter. We limit the number of possible values based on our experience so as to keep the model training time reasonable and decrease the time cost of the model training process.

Table 2.6: Tunable SML algorithm parameters and the corresponding value range

SML algorithm	**Tuning param**	**Description**	**Value ranges**
ANN	size	Number of units in the hidden layer	5, 7, 9, 11, 13, 15, 17, 19
	decay	Weight decay as a regularization to avoid the over-fitting of the function	0.1, 0.2, 0.3, 0.4, 0.5
KCCA	sigma	Inverse kernel width for the Radial Basis kernel function	0.05, 0.1, 0.5
MARS	degree	Maximum degree of interaction of the hinge functions	1, 2, 3
SVM	C	Penalty coefficient in control of the fitting function (over-fitting & under fitting)	0.25, 0.5, 1
	sigma	Inverse kernel width for the Radial Basis kernel function	0.01, 0.1, 0.5

2.5 Results

In this section we present results comparing the performance of different SML algorithms. We first evaluate the impact of the feature reduction procedure in selecting the significant micro-architectural parameters, thereby reducing the total design space. We then compare the accuracy of the different SML algorithms in modeling the micro-architectural power and performance for different benchmarks. Our results indicate that MARS has the highest accuracy using the least number of architectural parameters as predictor variables.

2.5.1 Impact of Feature Reduction and Algorithm Parameters Tuning

Table 2.7 shows the values of the different SML power and performance model parameters after the tuning step for each benchmark. Tables 2.8 to 2.15 present results

Table 2.7: Summary of the tuned values of the SML model parameters

Benchmark	**Metric**	**ANN**		**KCCA**	**MARS**	**SVM**	
		size	**decay**	**sigma**	**degree**	**C**	**sigma**
x264	FPS	19	0.1	0.1	3	1	0.1
	Power	19	0.1	0.1	3	0.25	0.01
Raytrace	FPS	17	0.2	0.05	3	0.25	0.05
	Power	19	0.3	0.1	3	0.25	0.05
Blackscholes & Swaptions	Runtime	13	0.2	0.1	2	0.5	0.01
	Power	15	0.2	0.05	3	1	0.05
Freqmine & Streamcluster	Runtime	9	0.1	0.05	3	1	0.05
	Power	13	0.1	0.05	2	0.5	0.01

Table 2.8: Summary of total number of selected micro-architectural predictor variables in performance and power modeling. The associated percent decrease in the number of predictor variables is shown in parenthesis.

Benchmark	**SML Algorithm**	**Performance Modeling**	**Power Modeling**
x264	ANN	24 (12.50)	15 (42.31)
	KCCA	14 (46.15)	15 (42.31)
	MARS	6 (76.92)	10 (61.54)
	SVM	14 (46.15)	6 (76.92)
Raytrace	ANN	21 (19.23)	13 (50.00)
	KCCA	13 (50.00)	8 (69.23)
	MARS	13 (50.00)	8 (69.23)
	SVM	13 (50.00)	10 (61.54)
Blackscholes & Swaptions	ANN	18 (30.77)	13 (50.00)
	KCCA	14 (46.15)	12 (53.85)
	MARS	11 (57.69)	10 (61.54)
	SVM	14 (46.15)	10 (61.54)
Freqmine & Streamcluster	ANN	19 (26.92)	8 (69.23)
	KCCA	15 (42.31)	8 (69.23)
	MARS	14 (46.15)	8 (69.23)
	SVM	17 (34.62)	12 (53.85)

Table 2.9: Feature reduction and algorithm parameter selection impact

Benchmarks	**Metrics**	**R^2 increased by %**			
		ANN	**KCCA**	**MARS**	**SVM**
x264	FPS	3.74	26.57	17.48	14.92
	Power	2.43	27.34	10.14	24.2
Raytrace	FPS	3.53	34.39	15.24	10.9
	Power	7.12	18.92	13.35	14.74
Blackscholes & Swaptions	Runtime	9.2	14.46	16.27	14.67
	Power	10.74	36.01	13.3	8.75
Freqmine & Streamcluster	Runtime	6.13	34.8	12.33	23.06
	Power	13.63	27.71	20.61	7.74

evaluating the impact of feature set reduction on the power and performance SML models for different SML benchmarks. Table 2.8 summarizes the number of micro-architectural model parameters used in each of the SML models. The associated percentage decrease in the feature space is indicated in parenthesis. We note that in each case MARS uses the least number of micro-architectural predictor variables while in general ANN uses the most. For example, in modeling the performance (FPS) of the x264 benchmark, ANN uses 4 times the number of parameters that MARS uses. This results in a corresponding decrease in the design space of 99.99%. Table 2.9 shows the percentage increase in R^2 metric as a result of the feature reduction step by comparing the accuracy of the model trained with the selected significant features to the model trained with the full feature set. We note that in most SML algorithms reducing the number of features results in a significant increase in model accuracy. For performance (power) models KCCA shows the most improvement at about 27% (27%) whereas ANN shows the least improvement at about 5% (8%). Table 2.10 to Table 2.13 show which of the micro-architectural features is selected by the different SML algorithms in the power and performance models for the different benchmarks.

Table 2.10: Significant micro-architectural parameters selected in x264 benchmark modeling (PE: performance, PO: power). 1 means the feature is selected and 0 means the feature is not selected

Micro-architectural parameter	**ANN**		**KCCA**		**MARS**		**SVM**	
	PE (24)	PO (15)	PE (14)	PO (15)	PE (6)	PO (10)	PE (14)	PO (6)
IntAlu	1	0	1	0	1	0	1	0
IntMultDiv	0	0	1	0	0	0	1	0
FpAlu	1	1	0	1	0	1	1	0
FpMultDiv	1	1	0	1	0	0	0	0
LocalHistBits	1	0	1	0	0	0	0	0
GlobalHistBits	1	1	0	0	0	0	0	0
PhysIntRegs	1	1	1	1	1	1	0	0
PhysFpRegs	1	0	0	0	0	0	1	0
SQEntries	1	1	0	1	0	0	0	0
LQEntries	1	1	0	1	0	0	0	0
IQEntries	1	1	1	1	1	0	1	0
ROBEntries	1	0	0	0	0	0	0	0
BTBEntries	1	0	0	0	0	0	1	0
RASSize	1	0	0	0	0	0	0	1
L1Size	1	1	1	1	1	1	0	0
L1I$Assoc	1	1	1	1	0	0	1	1
L1D$Assoc	1	1	1	1	0	1	1	0
DTLBSize	1	1	0	1	0	0	0	0
ITLBSize	1	0	1	0	0	0	1	0
CoreFreq	1	0	1	1	1	1	1	1
L2Size	1	1	0	1	0	1	0	1
L2Assoc	1	1	1	0	0	1	1	1
BSBFreq	1	0	1	1	0	1	1	1
FSBFreq	1	0	1	1	0	1	1	0
BusWidth	1	1	1	1	1	1	1	0

The impact of each micro-architectural feature in power and performance modeling can be judged by the number of models that selected the design parameter as a predictor variable. From Table 2.14 we conclude that for performance models, the integer ALU, number of integer registers, L2 and L1 cache associativity, bus width and choice of core frequency are important for all benchmarks. Other parameters such as floating point ALU is less significant for x264 whereas the L1 cache size is important for the x264

Table 2.11: Significant micro-architectural parameters selected in Raytrace benchmark modeling (PE: performance, PO: power). 1 means the feature is selected and 0 means the feature is not selected

Micro-architectural parameter	**ANN**		**KCCA**		**MARS**		**SVM**	
	PE (21)	PO (13)	PE (13)	PO (8)	PE (13)	PO (8)	PE (13)	PO (10)
IntAlu	1	1	1	0	1	0	1	0
IntMultDiv	1	1	0	0	0	0	0	0
FpAlu	1	1	1	1	1	1	1	1
FpMultDiv	1	1	0	0	1	0	0	0
LocalHistBits	0	0	0	0	0	0	0	0
GlobalHistBits	0	1	1	0	0	0	0	0
PhysIntRegs	1	1	0	0	1	0	0	1
PhysFpRegs	1	0	0	0	1	0	1	1
SQEntries	0	0	1	1	0	0	1	0
LQEntries	1	0	0	0	0	0	1	0
IQEntries	1	1	1	0	1	0	1	0
ROBEntries	1	0	1	0	0	0	0	0
BTBEntries	1	0	1	0	0	0	1	0
RASSize	1	0	1	0	0	1	0	0
L1Size	1	1	0	1	0	0	0	1
L1I$Assoc	1	0	0	1	1	1	1	1
L1D$Assoc	1	1	1	1	1	0	1	1
DTLBSize	1	0	0	0	1	0	0	0
ITLBSize	0	1	1	0	1	1	0	0
CoreFreq	1	1	0	0	1	1	1	1
L2Size	1	1	0	1	0	0	0	1
L2Assoc	1	0	1	1	1	1	0	0
BSBFreq	1	0	1	0	1	1	1	0
FSBFreq	1	0	1	0	0	1	1	1
BusWidth	1	1	0	1	1	0	1	1

benchmark. Regarding power models, from Table 2.15 we conclude that the L1 and L2 cache size is important for all benchmarks while the integer ALU is significant for x264 whereas floating point ALU is not significant for the Freqmine/Streamcluster benchmark. From Table 2.14 and 2.15 we note the possibility that depending on the benchmark that is executing some of the micro-architectural can be powered down.

Table 2.12: Significant micro-architectural parameters selected in Blackscholes/Swaptions benchmark modeling (PE: performance, PO: power). 1 means the feature is selected and 0 means the feature is not selected

Micro-architectural parameter	**ANN**		**KCCA**		**MARS**		**SVM**	
	PE (18)	PO (13)	PE (14)	PO (12)	PE (11)	PO (10)	PE (14)	PO (10)
IntAlu	1	0	0	0	1	0	1	0
IntMultDiv	1	0	0	0	0	0	0	0
FpAlu	1	1	1	1	1	1	1	1
FpMultDiv	1	1	0	1	1	1	1	1
LocalHistBits	0	0	1	0	0	0	1	0
GlobalHistBits	0	0	1	1	0	0	0	0
PhysIntRegs	1	1	1	0	1	1	1	1
PhysFpRegs	1	1	1	1	1	0	1	0
SQEntries	1	0	0	0	0	0	0	0
LQEntries	0	1	0	0	1	0	1	0
IQEntries	1	1	0	1	0	0	1	0
ROBEntries	0	0	1	0	0	0	0	0
BTBEntries	0	0	1	0	0	0	0	0
RASSize	0	0	1	1	0	0	1	0
L1Size	0	1	0	1	0	1	0	1
L1I$Assoc	1	0	1	1	0	0	0	0
L1D$Assoc	1	0	0	1	0	1	1	1
DTLBSize	1	0	1	0	0	0	0	0
ITLBSize	1	0	1	0	0	0	0	0
CoreFreq	1	1	1	1	1	1	1	1
L2Size	0	1	0	1	0	1	0	1
L2Assoc	1	0	1	1	1	0	1	0
BSBFreq	1	1	0	0	1	1	1	1
FSBFreq	1	1	0	0	1	1	0	1
BusWidth	1	1	1	0	1	1	1	1

Table 2.13: Significant micro-architectural parameters selected in Freqmine/Streamcluster benchmark modeling (PE: performance, PO: power). 1 means the feature is selected and 0 means the feature is not selected

Micro-architectural parameter	**ANN**		**KCCA**		**MARS**		**SVM**	
	PE (19)	PO (8)	PE (15)	PO (8)	PE (14)	PO (8)	PE (17)	PO (12)
IntAlu	1	0	1	0	1	0	0	0
IntMultDiv	0	0	0	0	0	0	0	0
FpAlu	1	0	1	0	1	0	1	1
FpMultDiv	0	0	0	0	0	0	1	0
LocalHistBits	1	0	0	0	0	0	0	0

Table 2.13 (cont'd)

GlobalHistBits	1	0	1	0	0	0	0	0
PhysIntRegs	1	0	1	0	1	1	1	1
PhysFpRegs	0	0	1	0	0	0	0	0
SQEntries	1	1	1	0	1	0	1	0
LQEntries	1	1	1	0	0	0	0	0
IQEntries	1	0	1	1	1	0	1	0
ROBEntries	0	0	1	0	0	0	1	1
BTBEntries	1	0	1	1	0	0	0	0
RASSize	1	0	1	0	1	0	1	0
L1Size	0	1	0	1	0	1	0	1
L1I$Assoc	1	0	0	0	1	0	1	1
L1D$Assoc	1	0	0	0	1	1	1	1
DTLBSize	1	1	0	1	0	0	1	0
ITLBSize	0	0	1	0	0	0	1	0
CoreFreq	1	1	1	1	1	1	1	1
L2Size	0	1	0	1	0	1	0	1
L2Assoc	1	1	1	1	1	0	1	1
BSBFreq	1	0	0	0	1	1	1	1
FSBFreq	1	0	0	0	1	1	1	1
BusWidth	1	1	1	1	1	1	1	1

Table 2.14: Significance of each micro-architectural design parameter determined by the number of SML models that uses it for performance modeling

Micro-architectural parameter	**x264**	**Raytrace**	**Blackscholes & Swaptions**	**Freqmine & Streamclusters**
IntAlu	4	4	3	3
IntMultDiv	2	1	1	0
FpAlu	2	4	4	4
FpMultDiv	1	2	3	1
LocalHistBits	2	0	2	1
GlobalHistBits	1	1	1	2
PhysIntRegs	3	2	4	4
PhysFpRegs	2	3	4	1
SQEntries	1	2	1	4
LQEntries	1	2	2	2
IQEntries	4	4	2	4
ROBEntries	1	2	1	2
BTBEntries	2	3	1	2
RASSize	1	2	2	4
L1Size	3	1	0	0

Table 2.14 (cont'd)

L1I$Assoc	3	3	2	3
L1D$Assoc	3	4	2	3
DTLBSize	1	2	2	2
ITLBSize	3	2	2	2
CoreFreq	4	3.	4	4
L2Size	1	1	0	0
L2Assoc	3	3	4	4
BSBFreq	3	4	3	3
FSBFreq	3	3	2	3
BusWidth	4	3	4	4

Table 2.15: Significance of each micro-architectural design parameter determined by the number of SML models that uses it for power modeling

Micro-architectural parameter	**x264**	**Raytrace**	**Blackscholes & Swaptions**	**Freqmine & Streamclusters**
IntAlu	4	1	0	0
IntMultDiv	0	1	0	0
FpAlu	3	4	4	1
FpMultDiv	2	1	4	0
LocalHistBits	0	0	0	0
GlobalHistBits	1	1	1	0
PhysIntRegs	3	2	3	2
PhysFpRegs	0	1	2	0
SQEntries	2	1	0	1
LQEntries	2	0	1	1
IQEntries	3	1	2	1
ROBEntries	0	0	0	1
BTBEntries	0	0	0	1
RASSize	0	0	1	0
L1Size	4	4	4	4
L1I$Assoc	2	2	1	1
L1D$Assoc	4	4	3	2
DTLBSize	2	0	0	2
ITLBSize	0	1	0	0
CoreFreq	2	3	4	4
L2Size	4	4	4	4
L2Assoc	3	1	1	3
BSBFreq	3	1	3	2
FSBFreq	3	2	3	2
BusWidth	3	4	3	4

2.5.2 SML Models Accuracy Analysis

To evaluate the accuracy of the SML models, we select another 50 design points from the same design space using the *MaxiMin* data sampling technique (See section 2.4.4).

Figure 2.5 show the prediction error distribution for the different SML performance and power models for all benchmarks. The RMSE and R^2 values for the performance and power models are shown in Tables 2.16 and 2.17 respectively. All performance (power) models show an $R^2 > 0.7$ (0.75) with an $R^2 > 0.9$ (0.9) for MARS. It should be noted that for all SML models, the training data set comprises of only 0.0000138% of the total design space.

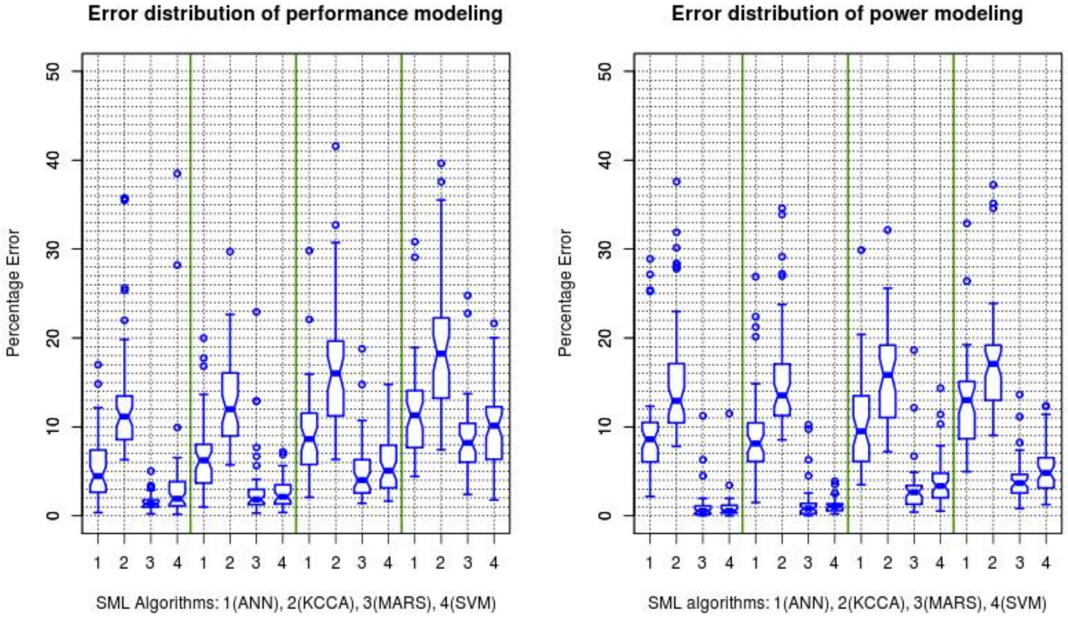

Figure 2.5: Prediction error distribution plot of performance and power models. The benchmarks are separated using the green solid line with from left to right: (1) x264, (2) Raytrace, (3) Blackscholes & Swaptions, (4) Freqmine & Streamcluster

Table 2.16: RMSE and R^2 for processor performance modeling

Benchmarks	**ANN**		**KCCA**		**MARS**		**SVM**	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
x264	1.0447	0.8651	1.8964	0.7798	0.2377	0.9525	0.2881	0.9337
Raytrace	1.1223	0.8513	1.9543	0.7644	0.2735	0.9452	0.3031	0.9265
Blackscholes & Swaptions	1.1576	0.8358	2.0846	0.7386	0.6435	0.8917	0.9813	0.8686
Freqmine & Streamcluster	1.1391	0.8472	1.9974	0.7523	0.5257	0.9076	0.7464	0.8876

Table 2.17: RMSE and R^2 for processor power modeling

Benchmarks	**ANN**		**KCCA**		**MARS**		**SVM**	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
x264	0.1594	0.8308	1.2235	0.7849	0.1307	0.9764	0.2435	0.9496
Raytrace	1.1894	0.8201	1.2848	0.7744	0.1515	0.9867	0.2889	0.9323
Blackscholes & Swaptions	1.2021	0.8001	1.6433	0.7518	0.5216	0.9006	0.7113	0.8912
Freqmine & Streamcluster	1.2181	0.8084	1.5151	0.7646	0.3171	0.9245	0.4849	0.9119

Table 2.18 compares the model feature training time for the different SML algorithms. All models are trained on an Intel Xeon workstation with a total of 8 cores. As seen in Table 2.18, MARS is significantly faster compared to the other SML algorithms.

Table 2.18: Average single threaded model training cost for each SML algorithm in modeling performance and power

SML algorithms	**Modeling cost (seconds)**	
	Performance	**Power**
ANN	640.26	431.42
KCCA	3926.38	3870.93
MARS	39.71	27.93
SVM	132.54	104.10

2.6 Design Space Exploration

We now utilize the SML power performance models to perform design space exploration of the target quad core architecture. Specifically, we seek to generate Pareto optimal power-performance fronts for the different benchmarks considered. In a multi-objective optimization problem, Pareto optimal solution describes a situation where no further optimization of any of the objectives is possible without sacrificing the other objectives.

We use the *NGPM* (NSGA-II library in Matlab) tool [44] to generate the Pareto optimal fronts. Non-dominated Sorting Genetic Algorithms-II (NSGA-II) algorithm is an evolutionary algorithm first described by [45]. More details on these algorithms are given in Chapter 4, Section 4.2.5.

Figures 2.6 to 2.9 gives the Pareto fronts for the different benchmarks considered in this work. As explained in Section 2.4.2, for x264 and Raytrace benchmarks, we use frames per second (FPS) as the performance metric, while for Blackscholes/Swaptions and

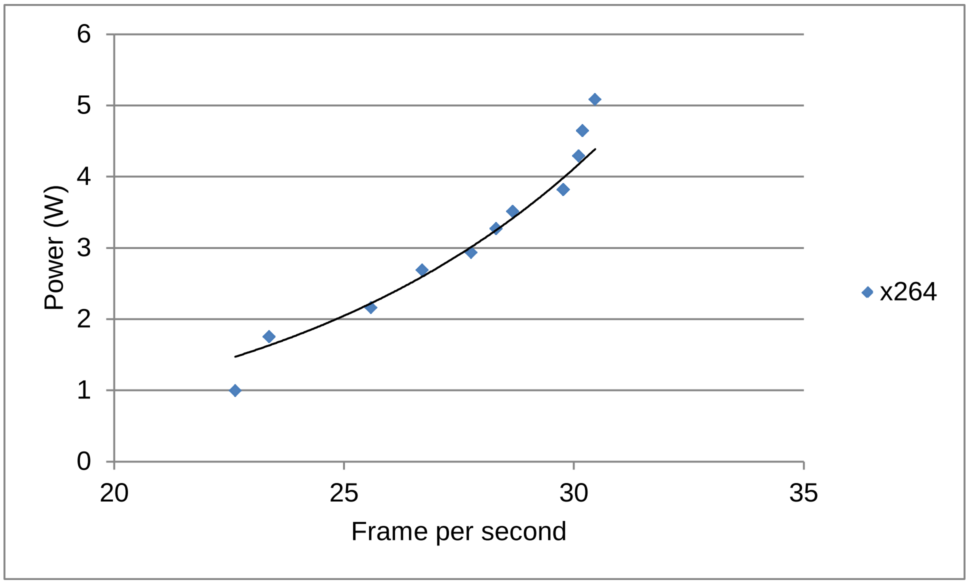

Figure 2.6: Pareto optimal power-performance front for x264

Freqmine/Streamcluster, we use the instructions per cycle (IPC) the performance metric. As expected for all benchmarks the power consumed increases with performance.

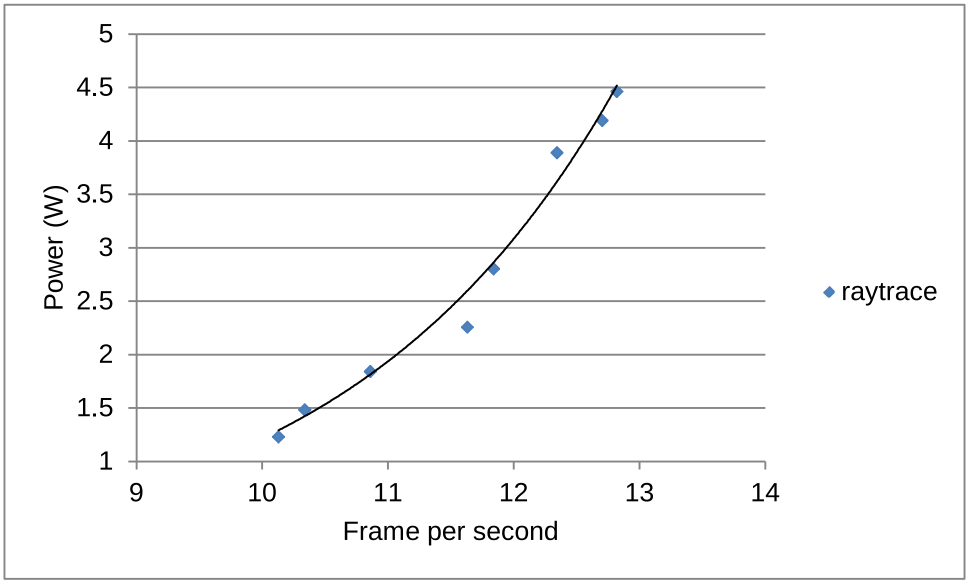

Figure 2.7: Pareto optimal power-performance front for Raytrace

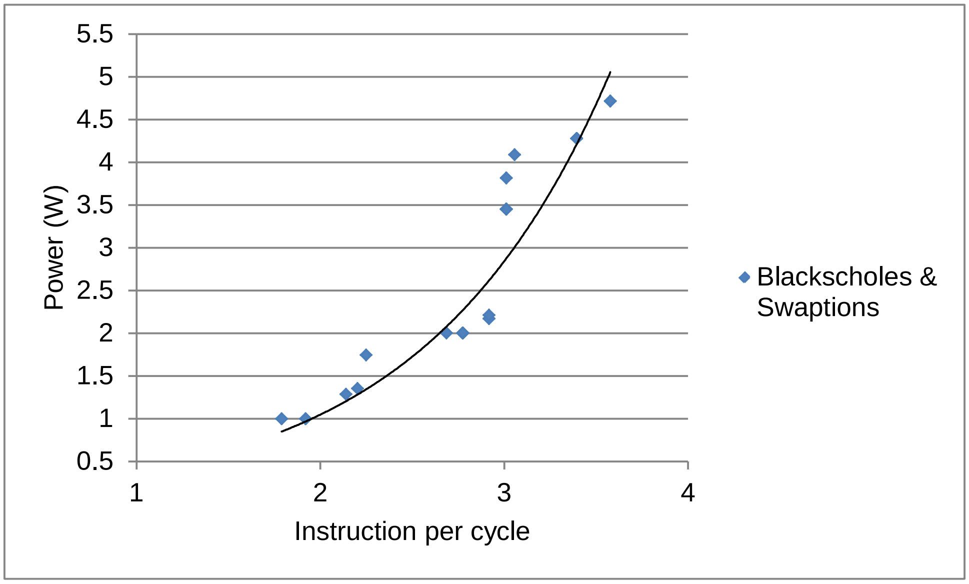

Figure 2.8: Pareto optimal power-performance front for Blackscholes/Swaptions

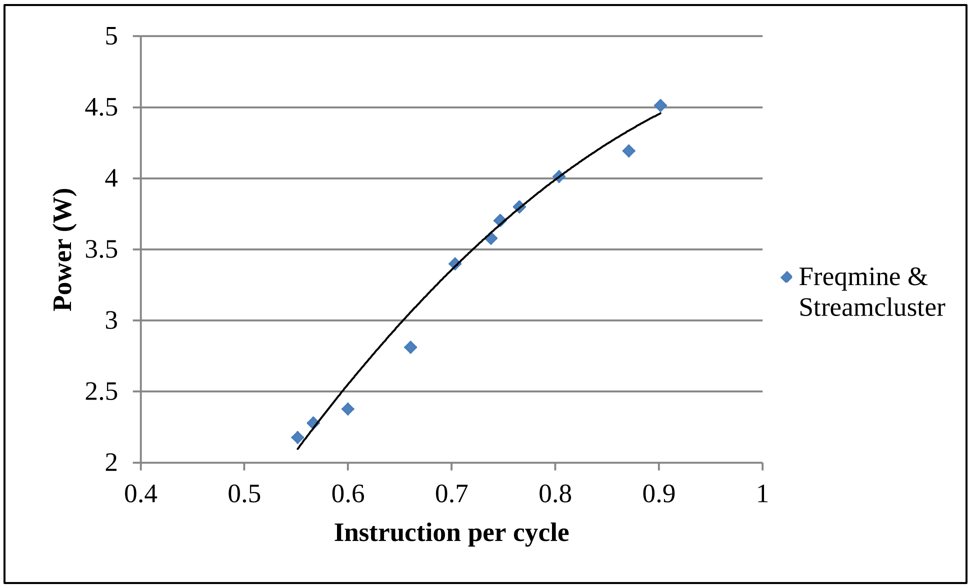

Figure 2.9: Pareto optimal power-performance front for Freqmine/Streamcluster

2.7 Related Work

Exploring the many-core processor design space through exhaustive cycle-accurate simulation is not practical due to the prohibitively long simulation time and its superliner increase as the numbers of cores are scaled. Several techniques have been proposed that avoids exhaustive simulations in effectively exploring the uniprocessor [46] – [49] and many-core [1] [2] [50] [51] design space. We first review recent research on modeling and exploring multi- and many-core architectures.

Lee et al. [1] minimize many-core simulation times in estimating performance through composable regression models for baseline uniprocessor performance, cache contention, and delay penalty. Their uni-core 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. [2] 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 [50] 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 include 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 [52] 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., [53] propose a sensor network benchmark, WiSeNBench, and use an ARM simulator to identify some of the key characteristic behaviors; Lin et al. [54] use a combination of analytical models and simulations to explore core-centric network processor architectures; Salehi et al., [55] 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. [56] present 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 uni-core processor architectures. Although, our methodology can incorporate compiler optimizations, these optimizations alone may not achieve sufficient performance on many-core processors.

In Dubach et al.'s another work [47], they present an architecture-centric approach. Their model is based on a simple linear combination of the design space of several individual programs from the training set. They could accurately predict the performance, energy or ED of a range of programs within a massive micro-architectural design space. Along with these progresses, they also suggest some characteristics of the design space, such as the register file has greatest impact on performance, the best configurations for cycles tend to have a wide pipeline and have a large reorder buffer. However, their work is still limited to uni-core processor architectures and our goal is different from theirs which is to tune the architecture joint benchmark.

Datta [51] presents a design exploration framework which employs an instruction trace-driven cycle-accurate simulator to accurately measure power and performance of embedded many-core heterogeneous processors based on a network IP packet processing embedded application. He then constructs a linear regression model to predict the power and CPI per thread of a given architecture configuration vector. He employs genetic algorithm to explore the design space to achieve the overall optimal configuration through a cost function of total power dissipation and real-time constraints.

2.8 Conclusions

In this dissertation, we investigate the suitability of a number of machine learning algorithm in micro-architectural modeling and concluded that the Multivariate Adaptive Regression Splines shows good performance both in terms of accuracy and model construction time. Although we have compared the performance of a number of machine learning algorithms, a study of other algorithms such as Kriging, Radial Basis Functions can be undertaken. In application areas in mechanical engineering [57] such algorithms have shown comparable performance to Multivariate Adaptive Regression Splines. We can also extend the application to a number of other benchmarks such as SPEC CPU2006 and EEMBC embedded microprocessor benchmarks. We can also investigate the suitability our model driven approach in the incorporation of higher level architectural parameters such as the type of the instruction set and compiler options.

CHAPTER 3: HIERARCHICAL MODELING FRAMEWORK FOR MANY-CORE PROCESSORS

3.1 Introduction

In this chapter, we present a hierarchical performance, power, and area modeling framework that can be used to rapidly explore the design space of many-core processors. Despite the vast reduction in design time possible with the statistical machine learning (SML) model driven micro-architectural exploration presented in Chapter 2, its scalability as the number of cores increase is limited. The primary reason for this limited scalability is the super-linear increase in simulation time required to generate the training data set. For example, for a 16 core shared memory many-core processor, performing a detailed simulation for a single training data point requires about 20 hours of simulation time on an 8 processor Intel Xeon workstation. Prior efforts in modeling many-core processors use either the non-scalable detailed simulation approach or employ coarse grained simulations with limited accuracy.

To achieve our goal of building scalable many-core models with good accuracy, we employ a “divide and conquer” strategy in constructing a hierarchical performance, area, and power models for many-core processors. The many-core processor is considered as clusters-of-cores sharing a memory through a global interconnect. Each cluster can have one or more cores, with private L1 caches, sharing the L2 cache through a local interconnect. The clusters may be homogeneous with identical interaction set architectures (ISAs) and microarchitectures, or may be heterogeneous. The cores in the clusters could be in-order processors, out-of-order processors, digital signal processors (DSPs), graphic processing units (GPUs), application specific processors (ASIPs) or non-programmable custom designed accelerators (ASICs). Each cluster is typically designed to run a single class of applications.

The entire micro-architectural design space of a many-core processor is divided into cluster-level parameters and interconnect-level parameters. Performance, power, and area models are constructed separately for the clusters and the global interconnect. The interactions between the clusters are captured through two parameters – the interconnect latency, and the average injection rate of packets into the network. For the cluster performance models, the interconnect latency is a predictor variable, while the average injection rate is a predicted output. For the interconnect performance models, the average injection rate is a predictor variable, while the interconnect latency is a predicted output. Note that the packet injection rate affects the interconnect access latency, while the interconnect access latency depends on the packet injection rate. An iterative approach involving the convergence of the values of these two parameters within an error threshold is used to ensure that the cluster and global interconnect models are consistent. The performance of the many-core processor is then the outputs of the individual cluster models. Alternatively, the overall many-core performance metric can be obtained through a weighted average of the outputs of the individual cluster performance models. Since the area and power are additive, the many-core area and power models are simply the sum of the cluster and global interconnect models.

We apply our proposed hierarchical modeling framework to six benchmarks from PARSEC: x264, Raytrace, Blackscholes, Swaptions, Freqmine and Streamcluster. The target architecture is a 64-core many-core processor with the cores based on the Alpha 21264 out-of-order processor. The cores have private L1 data and instruction cache, and 4 cores are clustered sharing a unified cache coherent L2 cache through a local bus based interconnect. The 16 cluster-of-cores share the main memory organized as memory banks through a global interconnect. The global interconnect is a 4x4 directory based 2D-mesh network with 16 memory controllers. Each cluster is an individual network node in the interconnect. We simulate the performance of the clusters using the Gem5 architectural simulator, the performance of the interconnect using the GARNET network simulator and estimate the power and area using the McPAT modeling framework targeting the 22nm technology node.

We present a comparison of the relative performance of the different statistical machine learning algorithms in modeling both the core and the interconnect. We then evaluate the accuracy of the many-core model against detailed simulations. We also present a scalability analysis of the modeling time using our proposed hierarchical modeling methodology. Finally, we use the many-core model to generate Pareto optimal performance-power fronts.

The rest of the chapter is organized as follows – in Section 3.2 we describe our hierarchical modeling framework in detail. We present the evaluation setup and the evaluation results in Section 3.3 and 3.4 respectively. In Section 3.5, we review related work on hierarchical modeling of many-core processors and in Section 3.6 we conclude the chapter identifying possible extensions of our work.

3.2 Hierarchical Performance, Power, and Area Modeling Framework

In Chapter 2, we constructed a statistical performance, area, and power model for a bus-based many-core processor with a small number of cores and validated the statistical machine learning based modeling strategy in chip multi-processor design. In this chapter, we extend our work to a many-core architecture with a large number of cores by utilizing a hierarchical modeling strategy. Figure 3.1 shows the organization of our many-core processor.

The many-core processor has multiple compute clusters sharing the main memory through a global interconnect. A cluster consists of one or more cores, the shared L2 cache, and a bus-based local interconnect. The clusters can be heterogeneous and each cluster is adapted to execute a specific class of applications. The hierarchical modeling framework utilizes 7 different component models constructed through simulation and SML modeling: individual cluster performance model, individual cluster power model, individual cluster area model, individual cluster injection rate model, global interconnect

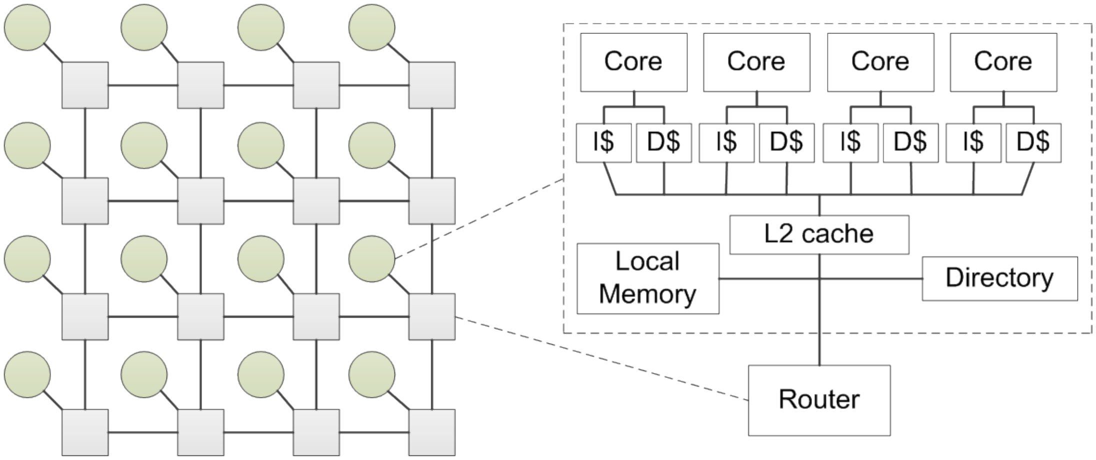

Figure 3.1: Our target 64-core processor with bus based local memory and mesh based global interconnect

latency model, global interconnect power model, and global interconnect area model. The predictor variables considered include core, cache, and local interconnect parameters as given in Table 3.1. Operating multiple clusters constituting a many-core processor can result in contention on the global interconnect thus affecting the latency from L2 cache to the main memory. We therefore include memory access latency as a predictor variable for modeling the performance of the cluster. The injection rate parameter models the average number of packets generated per cycle due to the highest level cache misses and write backs. The interconnect access latency determines the injection rate, and vice versa.

Figure 3.2 shows the proposed hierarchical many-core processor performance, power, and area modeling framework. In this framework, we assume the 7 individual models described above are already obtained through simulation and modeling using techniques described in Chapter 2. We then synthesize the many-core model as follows - given the micro-architectural design parameters and benchmarks, we attempt to find the injection rate and the corresponding network latency. We first choose a random initial network latency and obtain the corresponding injection rate through the cluster injection rate model. The new injection rate is then input to the interconnect model to calculate the new network latency. This process continues until the difference between the new network latency and the current network latency is below a certain threshold. The performance of the many-core processor is obtained from the individual cluster models, while the power and area of the entire many-core processor is the sum of the individual cluster and interconnects power and area. All individual models are evaluated using the network latency and injection rate obtained above.

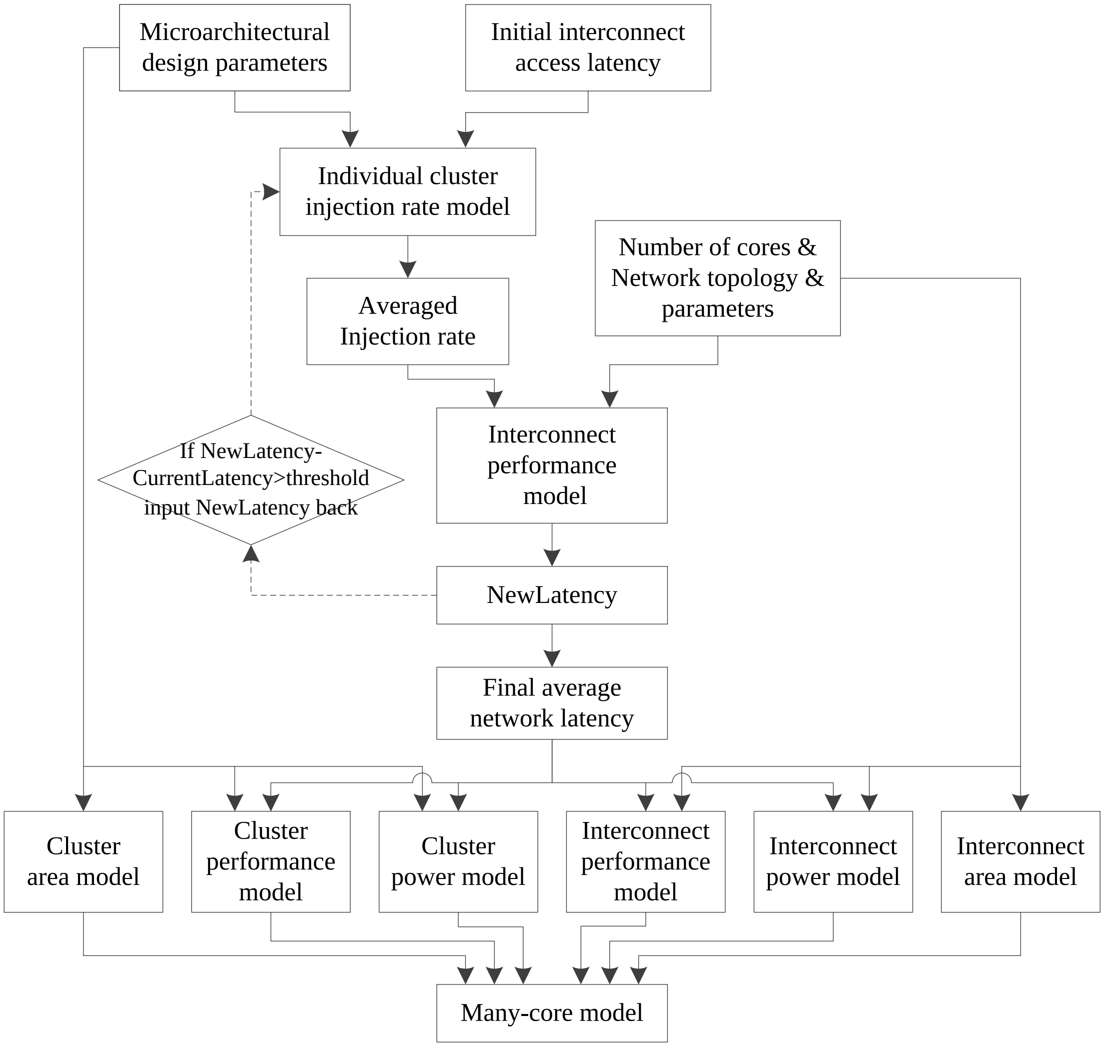
Microarchitectural design parameters
Initial interconnect access latency
Individual cluster injection rate model
Averaged Injection rate
Number of cores & Network topology & parameters
If NewLatency-CurrentLatency>threshold input NewLatency back
Interconnect performance model
NewLatency
Final average network latency
Cluster area model
Cluster performance model
Cluster power model
Interconnect performance model
Interconnect power model
Interconnect area model
Many-core model

Figure 3.2: Performance, power and area hierarchical modeling framework for many-core processors

3.3 Evaluation

3.3.1 Target Architecture and Design Space

We seek to model a 64-core many-core architecture composed of cluster of quad-core processors, with each core having a private L1 cache, 4 cores sharing an L2 cache, and all 64 cores sharing the main memory. The individual cores are out-of-order and are based on the Alpha 21264 processor. Each out-of-order core is a single threaded four issue processor with split instruction and data caches. The L1 and L2 caches are non-blocking cache with miss status holding register (MSHR) and write buffers (WB) for read and write misses. Cache coherency is maintained among the 4 cores sharing the L2 cache. The cache replacement policy is LRU and the cache coherence protocol is bus-based MOESI snooping protocol. Our local memory system is a classic bus-based model and the bus arbitration follows first-come-first-serve logic, and uses round-robin scheduling for bus accesses. The global interconnect is a 4x4 directory based 2D-mesh network with 16 memory controllers. Table 3.1 lists the cluster level model predictor variables and their possible value ranges.

Table 3.1: Cluster-level micro-architectural design parameters

Parameters	**Value ranges**
Number of integer Alu-MultDiv	2-1, 4-1, 4-2
Number of floating point Alu-MultDiv	1-1, 2-1, 4-1
Number of physical integer registers	64, 128, 196
Number of physical floating point registers	64, 128, 196
Number of store-load queue entries	32-48, 32-64, 48-64
Number of instruction queue entries	16, 32
Return address stack size	16, 32
L1 cache size (KB)	32, 64
L1 Instruction cache associativity	4, 8
L1 Data cache associativity	4, 8
DTLB size	32, 64, 128
ITLB size	32, 64, 128
Core frequency (GHz)	0.8, 1.0, 1.33, 1.8, 2.0, 2.33, 2.66
L2 cache size (MB)	1, 2, 4
L2 cache associativity	8, 16
Back side bus frequency	Core frequency
Bus Width (Byte)	8, 16, 24
Latency (Cycles)	20, 25, 30, 35, 40

The router in the 2D-mesh interconnect is a four-stage pipeline virtual channel router. The four stages are: buffer write and router compute (BWRC), virtual channel

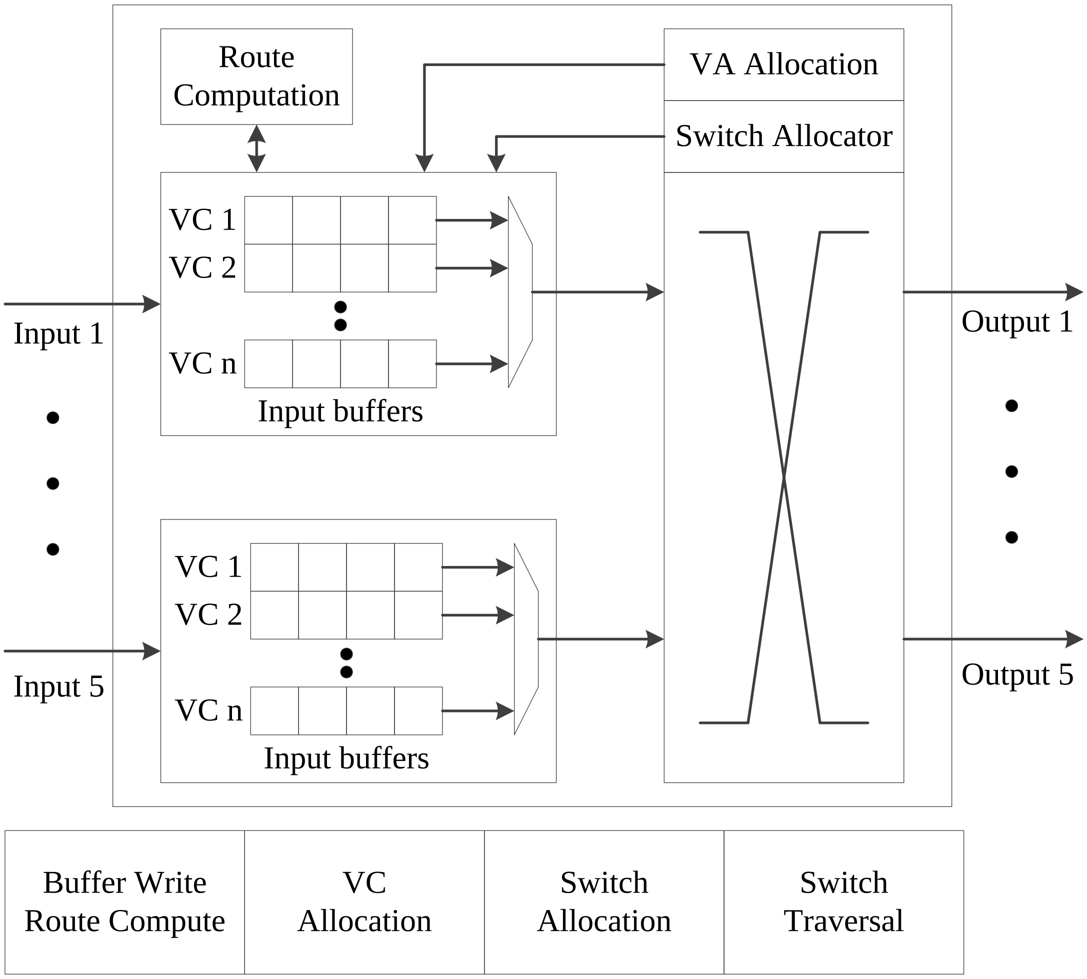

Figure 3.3: Four-stage pipelined virtual channel router

allocation (VA), switch allocation (SA) and switch traversal (ST). The network message is first broken into multiple packets, and each packet has header information that allows the receiver to reconstruct the original message. In a network communication scheme, each packet can be further broken into flits, the smallest unit of flow control. A packet consists of a head flit, followed by body flits, and ends with a tail flit. [58]. In BWRC, the incoming flits are buffered and their output ports are computed. In VA, the buffered flits are allocated in free virtual channels. In SA, the flit that has the right to go through the

Table 3.2: Interconnect parameters and parameter value range

Parameters	**Value Ranges**
Number of network nodes (processors)	16, 64
Injection rate (flit/node/cycle)	0.01 to 0.7 in increments of 0.025
Flit size (Byte)	2, 4, 8
Virtual channels per virtual networks	2, 4, 8, 12, 16
Number of buffers per virtual channel	2, 3, 4, 5, 6, 7, 8

crossbar is determined. In ST, the flit that is chosen by the SA traverses the cross switch. Figure 3.3 depicts the pipeline stages. Table 3.2 lists the cluster level model predictor variables and their possible value ranges.

3.3.2 Benchmarks and Data Sampling

We use six benchmarks from Parsec: x264, Raytrace, Blackscholes, Swaptions, Freqmine and Streamcluster. The benchmarks are described in Table 2.2 and the input problem size of each benchmark is listed in Table 2.3. The media benchmarks x264, and the graphics benchmark Raytrace are run on individual clusters. The finance benchmarks Blackscholes and Swaptions are run on a single cluster with 2 cores per application. Similarly, the data mining benchmarks Freqmine and Streamcluster are also run a single cluster with 2 cores per application.

In addition to the 1054 training data points collected in Chapter 2, we sample another 40 design points from the training data through random selection of micro-architectural configurations with different memory access latencies. This new 1094 samples training data set is used to model the cluster performance, power, and injection rate. To model the interconnect access latency, we do an LHS sampling of 300 training data points obtained using the GARNET interconnect simulator.

3.3.3 Simulation Tools

3.3.3.1 Cluster-level performance modeling

We use the Gem5 simulator [20] for the performance modeling of our target system. Gem5 is an event-driven cycle-accurate simulator, which provides a highly configurable simulation framework, multiple ISAs, diverse CPU models, multiple cache coherence protocols and interconnects models. It also supports both full-system and system-call emulation modes. The full-system simulation mode simulates a complete computer system including operating system kernel and I/O devices, while in system-call emulation mode, the common system calls are emulated by calling the host OS. Besides, it provides a clear interface for check-pointing, fast-forwarding, debugging and statistics.

Our Alpha ISA based architecture described in Section 2.1 is simulated in full-system mode, which models a complete Linux system with kernel version 2.6.27. To construct the cluster performance model, power model and area model, we run the benchmarks described in Section 3.3.2 on Gem5 by fast forwarding the simulation starting point to the pre-defined start of region of interest (ROI) where parallel execution commences. We collect statistics such as the number of integer/floating point instructions, the number of accesses and misses to each hardware unit and the total number of cycles when the benchmark is running in the ROI.

3.3.3.2 Cluster-level power and area modeling

We use McPAT 0.8 [28] for the power and area modeling of our target architecture. McPAT is an integrated power, area, and timing hierarchical modeling framework for multi-core and many-core processor configurations targeting technologies ranging from 90nm to 22nm. McPAT supports both static and dynamic power modeling at the micro-architectural level, circuit and technology level.

At the micro-architectural level, McPAT includes models of major architectural components such as cores, interconnects, caches, memory controllers, and clocking. At the circuit level, the architectural building blocks are mapped into four basic circuit structures: hierarchical wires, arrays, complex logic, and clocking networks. At the technology level, the physical parameters of devices and wires, such as unit resistance, capacitance, and current densities, is calculated based on the data from the ITRS roadmap. In our work, we target the 22nm technology node.

3.3.3.3 Interconnect performance, power and area modeling

We use GARNET and McPAT to model the performance, power, and area of the interconnect targeting the 22nm technology node. The interconnect is modeled with the four-stage pipeline routers described in Section 3.3.1. In every cycle, each node performs a Bernoulli trial with probabilities equal to the injection rate to generate new packets. The destination directory of the generated packet is chosen randomly from all the directories. The generated packets are randomly tagged as Load, Instruction Fetch (InstFetch) or Store packets. The Load and InstFetch packets are injected as 8-byte control packets and the Store packet is injected as a 72-byte data packet.

Table 3.3 shows the average number of packets generated with the problem size listed in Table 2.3. The average number of packets is the sum of the L2 cache overall misses and L2 write-backs collected from the detailed cycle-accurate simulations of the cluster for each benchmark.

Table 3.3: Summary of the simulated benchmarks with average packets generated

Benchmark	Average packets
x264	1.65 M
Raytrace	0.84 M
Blackscholes & Swaptions	1.96 M
Freqmine & Streamcluster	4.52 M

3.3.4 Statistical Machine Learning Algorithms

We provide a brief of the machine learning algorithm used in building our hierarchal many-core model. A more detailed description is given in Chapter 2, section 2.3.

3.3.4.1 Artificial neural networks

ANN is a non-linear statistical data model inspired by biological nervous systems and neural networks. The representational power of ANN is rich enough to express complex interactions among variables – any function can be approximated to arbitrary precision by a three-layer ANN [16].

Three types of parameters define an ANN: the interconnection pattern, the learning process, and the activation function [16]. Figure 3.4 shows a feed-forward ANN and a sigmoid activation function. The feed forward network contains three layers with P units in the input layer, H units in the hidden layer, and L units in the output layer. The sigmoid activation function is applied to the units in the hidden layer and output layer. ω_{hij} represents the weight between input unit $\{x_i\}_{i=1}^{P}$ and hidden unit $\{h_j\}_{j=1}^{H}$, and ω_{ojk} represents the weight between hidden unit $\{h_j\}_{j=1}^{H}$ and output unit $\{y_k\}_{k=1}^{L}$. To train the network, we choose the stochastic gradient descent version of the BackPropagation (BP) algorithm [16]. The BP algorithm is the most commonly used ANN learning technique. It

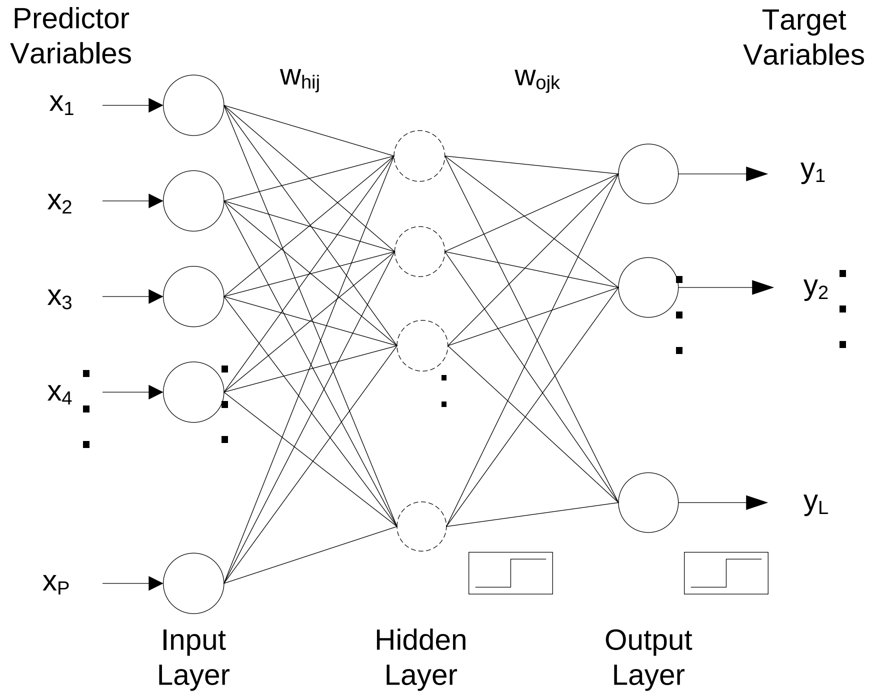

Figure 3.4: Example of a feed-forward ANN [16]

is composed of three parts: forward propagation, back propagation and weight update. In each iteration, the training input is first forwarded through the network to generate the output of each layer using the sigmoid function. Then, the errors are propagated backward through the network to calculate the error term of each unit. The weights in each layer are then updated using the gradient descent algorithm. This training process stops when the error difference between adjacent iterations is below a desired error threshold or a maximum number of iterations are reached.

3.3.4.2 Multivariate adaptive regression spline

Linear regression is the most widely used model to construct the approximation function $\hat{f}$ for each target variable using a linear relationship:

$$y = \hat{f}(\boldsymbol{x}) + \varepsilon = b + \sum_{i=1}^{P} \omega_i x_i + \varepsilon \tag{3.1}$$

Here y is the target variable of interest, b is an intercept term, ε is an error term, and ω_i is the corresponding coefficient of predictor variable x_i. A regression spline extends linear regression by using two or more polynomial fittings in each of the $K + 1$ disjoint regions separated by K points (knots). Multivariate adaptive regression spline builds on regression splines through the use of truncated power basis functions [14] and adaptive knot selection strategy [10] to construct the approximation function $\hat{f}$:

$$\hat{f}(\boldsymbol{x}) = \sum_{m=0}^{M} \omega_m B_m(\boldsymbol{x}) \tag{3.2}$$

Here M is the number of basis functions included in the model. One representation of these basis functions is:

$$B_m(\boldsymbol{x}) = \prod_{i=1}^{I_m} \left[s_{(i,m)} \left(x_{j_{(i,m)}} - t_{(i,m)} \right) \right]_+^Q \tag{3.3}$$

Here I_m is the number of factors (interaction order) of the m^{th} basis function and (i, m) denotes the m^{th} basis function and the i^{th} interaction order. $s_{(i,m)} = \pm 1$ indicates the positive (right) or negative (left) of the truncated power function pairs of the mth basis function. $x_{j_{(i,m)}}$ is one of the predictor variable $\{x_j\}_{j=1}^{P}$ and $t_{(i,m)}$ is the knot location.

The MARS algorithm employs a two-step procedure to build the model: the forward pass and backward pass. In the forward pass, the algorithm starts with one basis function which is just an intercept term, and then repeatedly adds new basis functions in pairs (two at a time) to the model. In the $(M + 1)^{th}$ iteration, the basis function which maximizes the reduction in SSE (sum of squared error) of the model built in the previous M iterations is selected, and here SSE is described by

$$SSE(M + 1) = \sum_{i=1}^{N} \left[y_i - \hat{f}_M(\boldsymbol{x}_i) - B_{(2M+1)}(\boldsymbol{x}_i) - B_{(2M+2)}(\boldsymbol{x}_i) \right]^2 \tag{3.4}$$

In Equation 3.3, all possible choices of the newly added basis function $B_{(2M+1)}(\boldsymbol{x}_i)$ and $B_{(2M+2)}(\boldsymbol{x}_i)$ are evaluated until the change of SSE of the model is below a given threshold or until the maximum number of basis functions is reached. At the end of the forward pass, the number of basis function is typically chosen to be substantially larger than would be optimal, and results in an over-fitted model. In the backward pass, the MARS algorithm prunes the model built in the forward pass by removing the least effective basis functions one at a time until only the intercept term is left. The least effective basis function is determined based on a modification of the Generalized Cross Validation (GCV) criterion [15] as described by:

$$GCV(M) = \frac{1}{N}\sum_{i=1}^{N} \frac{\left[y_i - \hat{f}_M(\boldsymbol{x}_i)\right]^2}{\left(1 - \frac{enp(M)}{N}\right)^2} \tag{3.5}$$

$$enp(M) = M + c * \frac{M-1}{2} \tag{3.6}$$

Here M is the number of basis functions in the model $\hat{f}_M(\boldsymbol{x})$ (including the intercept term), enp is the effective number of parameters given by Equation 3.6, *c* is the GCV penalty per knot and $(M-1)/2$ is the number of knots. The final output model is the one with the lowest GCV value. The numerator of Equation 3.5 is the MSE on the training data and the denominator represents a penalty for increasing model complexity (number of knots).

In our work, we use the implementation in the *caret* R package [43].

3.3.4.3 Kernel canonical correlation analysis

Canonical correlation analysis (CCA) is a SML algorithm to find a linear combination of two sets of variables $\boldsymbol{x}: \{x_1 .. x_P\}$ and $\boldsymbol{y}: \{y_1 .. y_L\}$ that has maximum correlation with each other. Given two column vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$, it tries to find vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ to maximize the correlation between $\boldsymbol{a^T X}$ and $\boldsymbol{b^T Y}$. However, CCA is only useful when the relationship of variables can be captured in linear form. For non-linear data sets, we use Kernel CCA (KCCA) which has the same objective as CCA but models non-linear variables through a kernel trick. The kernel trick tries to project the two datasets to a higher dimension and find the linear relationship between the projected two datasets. The first step is to collect N training samples using a sampling strategy such as LHS or random sampling. We then use the Gaussian kernel function to form an $N \times N$ parameter similarity matrix $\boldsymbol{K_x}$ whose $(i,j)th$ entry measures the similarity between vectors $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ in $\boldsymbol{x}$, and an $N \times N$ performance similarity matrix $\boldsymbol{K_y}$ whose $(i,j)th$ entry is the similarity between vectors $(\boldsymbol{y_i}, \boldsymbol{y_j})$ in $\boldsymbol{y}$. In the Gaussian kernel function, the Euclidean distance is used.

We input these two matrices into a generalized Eigen equation to project the similarity matrix $\boldsymbol{K_x}$ to a basis vector space $\boldsymbol{\alpha}$ by $\boldsymbol{K_x} \times \boldsymbol{\alpha}$ and the similarity matrix $\boldsymbol{K_y}$ to a basis vector space $\boldsymbol{\beta}$ by $\boldsymbol{K_y} \times \boldsymbol{\beta}$. The basis vector space $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are got from the N largest Eigen values. The projections of $\boldsymbol{K_x}$ on to basis vector space $\boldsymbol{\alpha}$ and the projections of $\boldsymbol{K_y}$ on to basis vector space $\boldsymbol{\beta}$ are mutually maximally correlated.

Once we get the projection of $\boldsymbol{X}$ on $\boldsymbol{\alpha}$, the prediction model is ready. For a new input configuration vector whose output we seek to predict, we use the same kernel function to get the similarity vector $\boldsymbol{V}$ between the new input configuration vector and all the vectors in the sample space. After this, we project $\boldsymbol{V}$ to basis vector space $\boldsymbol{\alpha}$ by $\boldsymbol{V} \times \boldsymbol{\alpha}$ and use a distance metric to find its *k* nearest neighbors in this space. We then average its neighbor's performance to obtain the predicted performance of the configuration we are interested.

3.3.4.4 Support vector machines

SVM is used to construct a hyperplane or set of hyperplanes in a high- or infinite-dimensional space which can be used for classification (SVC) or regression (SVR). Consider a set of training data: $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_N, y_N)\}$, where $\boldsymbol{x}_i$ is a set of predictor variables $\{x_{j_i}\}_{j=1}^{P}$ and y_i is the target variable in training data $\{i\}_{i=1}^{N}$. In SVC, the target variable y_i is a class label and the hyperplane is a separation of different classes. The goal of SVC is to find a hyperplane that has the largest distance to the nearest training data point of any class. Support Vectors (SVs) are the nearest training data points of any class to the hyperplane, and margin is the perpendicular distance between the hyperplane and SVs. With a larger margin, the hyperplane has a higher probability to classify a new data point correctly. So the optimization problem of SVC is to maximize the margin subject to the constraints of correctly classifying all the training data. In SVR, the target variable y_i is a real value. The goal of SVR is to find a hyperplane (approximating function $\hat{f}$) that has at most ϵ deviation from true value y_i of the target variable for all the training data. For SVR, there exist two common approaches: ϵ-SVR and nu-SVR. In ϵ-SVR, ϵ is a pre-defined acceptable amount of deviation between $\hat{y}$ and y of the target variable. While in nu-SVR, ϵ is determined as a part of the learning algorithm. It has been demonstrated in [17] [18] that although ϵ-SVR is simpler, it has an equal or even better performance than nu-SVR. So in our work, we consider ϵ-SVR. In ϵ-SVR, the margin is defined as the perpendicular distance between hyperplane $\hat{f}$ and $\hat{f} \pm \epsilon$. Any data point in this margin is considered as well fitted by $\hat{f}$. So with a larger margin, the hyperplane ($\hat{f}$) has a higher probability to predict y of a new data point within an acceptable ϵ deviation.

So the optimization problem of ϵ-SVR is to maximize the margin subject to the constraints of maximum ϵ deviation for all the training data.

3.4 Results

In this section we first present results on the accuracy of the selected SML algorithm based models on modeling the injection rate of the cluster and the average interconnect access latency. We then present results on the accuracy of the many-core model generated through the hierarchical modeling framework by comparing the predicted performance, power and area to those obtained by through detailed simulation of the entire processor. In evaluating the hierarchical modeling framework, the best SML algorithm for cluster performance, power, injection rate modeling, and interconnect access latency modeling is used. We also analyze the scalability of our hierarchical modeling framework with the number of cores and compare to the previously proposed hierarchical many-core processor design frameworks.

3.4.1 Injection Rate Model and Interconnect Model Accuracy

In this experiment, 50 validation points are sampled to evaluate the cluster injection rate model and global interconnect model using the LHS data sampling technique and simulated with Gem5, Garnet, and McPAT.

Figure 3.5 shows the prediction error distribution for the different SML injection rate and latency models for all benchmarks. The RMSE and R^2 values for the injection rate and network latency models are shown in Tables 3.4 and 3.5 respectively. All injection rate (latency) models show an $R^2 > 0.79$ (0.86) with an $R^2 > 0.92$ (0.95) for MARS. Figure 3.6 shows the prediction error distribution for the interconnect power and area models. The RMSE and R^2 values for the interconnect power and area models are shown in Table 3.6. All interconnect power (area) models show an $R^2 > 0.83$ (0.85) with an $R^2 > 0.92$ (0.94) for MARS. It should be noted that for cluster and interconnect models, the training data set comprises of only 0.001% and 5% of the cluster and interconnect design space.

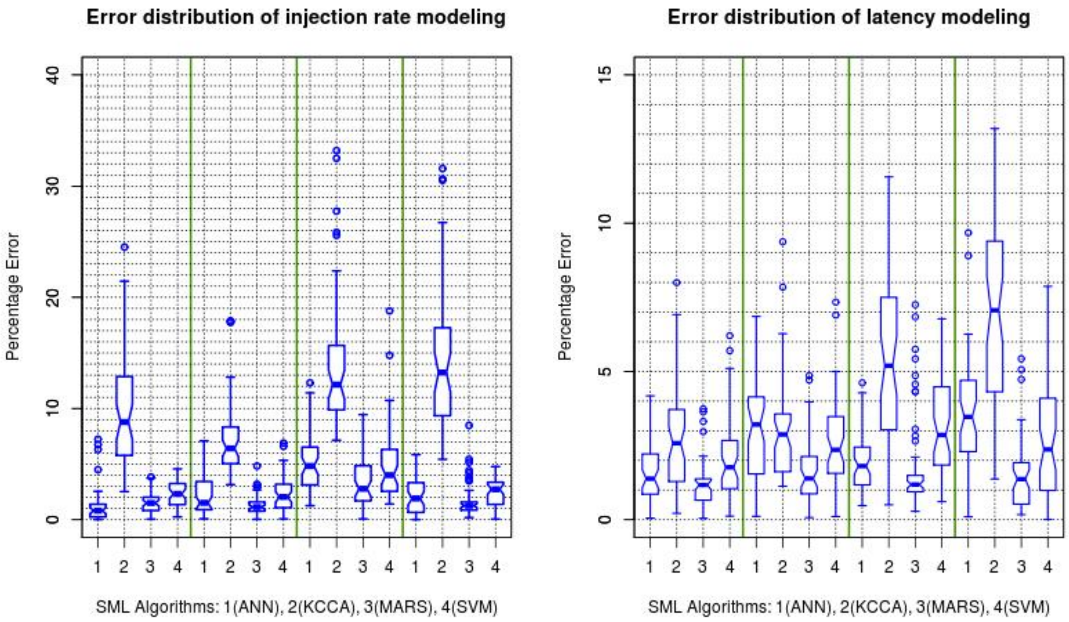

Figure 3.5: Prediction error distribution plot of injection rate and interconnect access latency models. The benchmarks are separated using the green solid line with from left to right: x264, Raytrace, Blackscholes & Swaptions, Freqmine & Streamcluster

Table 3.4: RMSE and R^2 of individual cluster injection rate modeling

Benchmarks	**ANN**		**KCCA**		**MARS**		**SVM**	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
x264	0.1296	0.9843	1.2113	0.8132	0.1332	0.9748	0.2012	0.9612
Raytrace	0.1357	0.9712	1.1274	0.8324	0.1301	0.982	0.1599	0.9623
Blackscholes & Swaptions	0.4864	0.9112	1.5126	0.8070	0.3173	0.9244	0.4961	0.9191
Freqmine & Streamcluster	0.1539	0.9656	1.2092	0.7979	0.1325	0.9776	0.2074	0.9542

Table 3.5: RMSE and R^2 of interconnect access latency modeling

Benchmarks	**ANN**		**KCCA**		**MARS**		**SVM**	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
x264	0.1325	0.9751	0.5029	0.9132	0.1254	0.9876	0.1312	0.9789
Raytrace	0.2887	0.9323	0.2817	0.9355	0.2313	0.9541	0.2832	0.9375
Blackscholes & Swaptions	0.2384	0.9498	0.7064	0.8952	0.1218	0.9886	0.2396	0.9446
Freqmine & Streamcluster	0.3206	0.9225	1.0233	0.8637	0.1297	0.9835	0.2384	0.9502

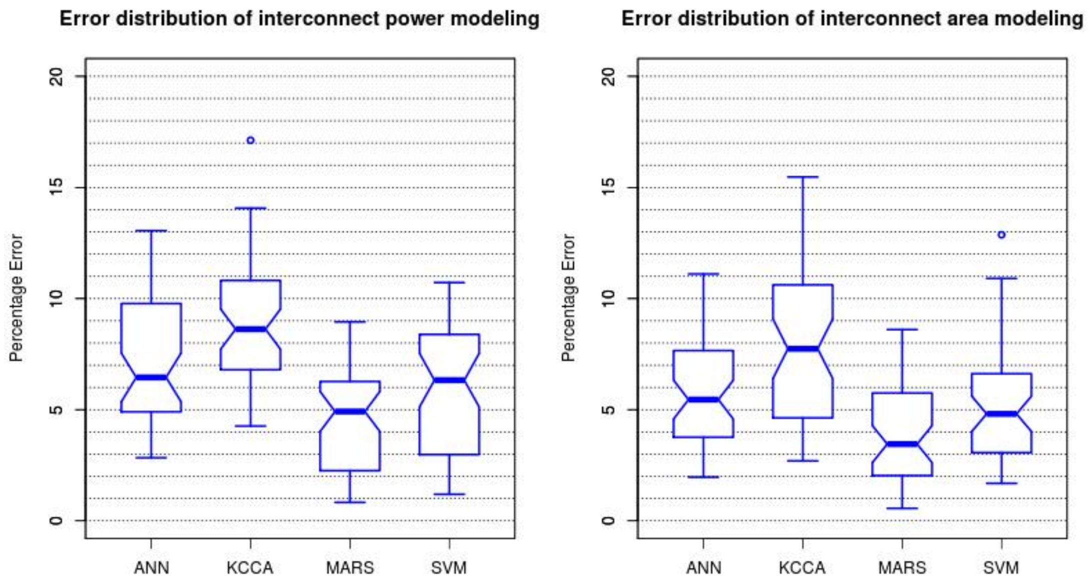

Figure 3.6: Prediction error distribution plot of interconnect power and area models.

Table 3.6: RMSE and R^2 of interconnect power and area models

Interconnect Metric	**ANN**		**KCCA**		**MARS**		**SVM**	
	RMSE	R^2	RMSE	R^2	RMSE	R^2	RMSE	R^2
Power	0.8165	0.8845	1.1253	0.8346	0.3277	0.9216	0.6978	0.9003
Area	0.6713	0.9082	1.0566	0.8501	0.2424	0.9411	0.5046	0.9120

3.4.2 Many-core Processor Model

As shown in Figure 3.2 the many-core processor model is synthesized from the cluster models and network latency models by iterating over the injection rate and network latency until 0.1% convergence threshold is reached. Figure 3.5 shows an example of the iteration procedure used to determine the network latency and injection rate. The network access latency is initialized to 20 cycles. As seen from Figure 3.7 the convergence occurs in 16 iterations. Table 3.7 shows the associated cluster and interconnects microarchitecture parameter values for different benchmark applications.

Table 3.7: Fixed cluster-level and interconnect-level parameter values

Parameters	**Selected value**			
	x264	**Raytrace**	**Black & Swap**	**Freq & Stream**
Number of integer Alu-MultDiv	4-1	4-1	4-2	4-1
Number of floating point Alu-MultDiv	2-1	2-1	4-1	2-1
Number of physical integer registers	128	128	196	128
Number of physical floating point registers	64	128	64	64
Number of store-load queue entries	32-48	32-48	48-64	32-64
Number of instruction queue entries	16, 32	32	16	32
Return address stack size	16, 32	32	32	16
L1 cache size (KB)	32, 64	32	64	64
L1 Instruction cache associativity	4	8	8	4
L1 Data cache associativity	8	8	4	8
DTLB size	64	64	32	128
ITLB size	32	32	64	32
Core frequency (GHz)	1.0	2.0	1.8	1.33
L2 cache size (MB)	1, 2, 4	2	2	4
L2 cache associativity	8, 16	16	8	8
Back side bus frequency	1.0	2.0	1.8	1.33
Front side bus frequency (MHz)	133	133	100	133
Bus Width (Byte)	8	8	16	16
Number of network nodes (processors)	16			
Flit size (Byte)	4			
Virtual channels per virtual networks	4			
Buffer size per virtual channel	4			

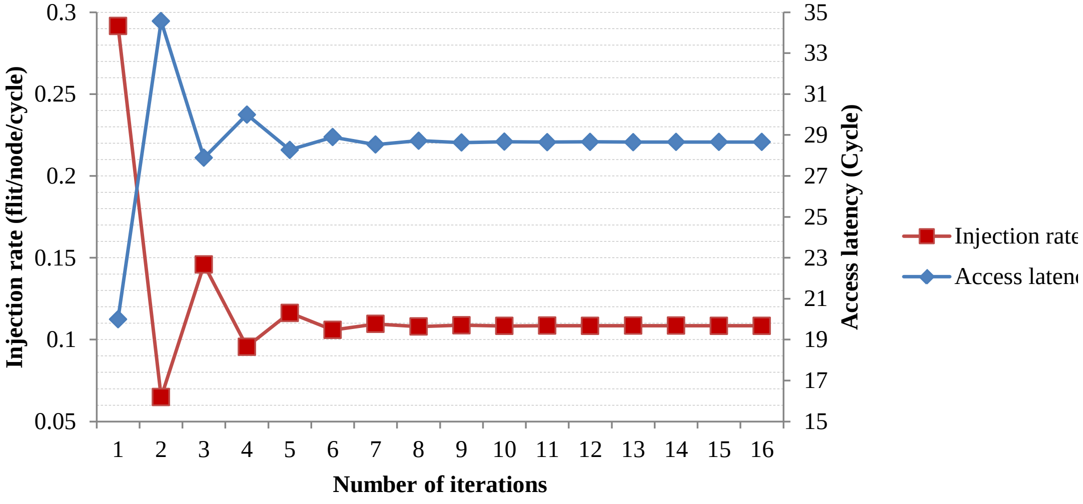

Figure 3.7: Convergence of injection and access latency

3.4.2.1 Evaluating many-core model accuracy

We compare the performance, power, and area results modeled using our hierarchical framework to those obtained using detailed simulation of the targeted many-core architecture. To keep our simulation times reasonable, we limit the number of clusters to be 4, with 4 cores per cluster for a total of 16 cores. The detailed simulation takes 38 – 42 hours for each design point depending on the benchmark. From the results of Chapter 2 for the cluster models, and section 3.4.1.1 for the interconnect models, MARS models are seen to be the most accurate. MARS is therefore used for all the modeling tasks.

In this experiment, each cluster executes a different benchmark. The statistics collection begins when all the clusters are running in its ROI session. Due to the different simulation time of the benchmarks and the different metrics in evaluating the performance of each benchmark, the x264 and Raytrace are set to finish in roughly the same simulation

Table 3.8: RMSE and R2 for performance and power with four clusters simultaneously running 6 different benchmarks

Benchmark	**Metric**	**R^2**	**RMSE**
x264	FPS	0.8229	1.1782
	Power	0.8414	1.1053
Raytrace	FPS	0.8633	1.0231
	Power	0.8896	0.7446
Blackscholes & Swaptions	CPI	0.8516	1.0984
	Power	0.8934	0.7095
Freqmine & Streamcluster	CPI	0.8674	1.0223
	Power	0.8795	1.0148

time by adjusting the corresponding problem size. Since x264 and Raytrace have long execution times, we rerun Freqmine & Streamcluster and Blackscholes & Swaptions to ensure that all 4 clusters are executing simultaneously. For validation, 16 design points (core + network micro-architectural parameters) are sampled using the LHS data sampling technique and simulated with Gem5 and McPAT.

The RMSE and R^2 values for the injection rate and network latency models are shown in Table 3.7. The performance models have an $R^2 > 0.82$ while the power models have an $R^2 > 0.84$.

3.4.3 Scalability Analysis

The SML based statistical modeling method accurately estimates the injection rate and the average interconnect access latency, and is validated through comparison to the detailed simulation of a 16-core processor. The efficiency (training cost) of our selected SML algorithms is shown in Table 3.8. As can be seen from the table, the training cost is very algorithm dependent. The minimum training cost can be nearly hundred times less than the maximum training cost. Please note that the model training cost also includes the algorithm parameter tuning time, which means that a smaller number of tuning parameters

Table 3.9: Model training cost for each SML algorithm in modeling performance, power, injection rate and average network latency

SML algorithms	**Modeling cost (seconds)**	
	Injection rate	**Access Latency**
ANN	212.18	41.07
KCCA	12.83	80.61
MARS	2.19	1.05
SVM	61.68	67.71

with few parameter value options could result in possibly less training time. Table 3.9 also shows that among the SML algorithms considered in our work, MARS is more than an order of magnitude times faster than other algorithms.

To analyze the scalability of our hierarchical modeling framework, and compare to other hierarchical methods proposed in in the literature, we develop a simple analytical model for the total training time. In our analysis we use the following notations – N is the number of training data points used to train the models, T_C is the detailed cycle-accurate simulation time per processing core or cluster, T_{Mesh} is the detailed cycle-accurate simulation time for the mesh interconnects, n is the total number of processing cores or clusters in the system. We employ an expression in [1], that is $T_n = n^r * T_C$, to represent the super-linear increment in the simulation time with the number of processing cores n. We observe a growth factor r in the range 1.22 to 1.31 in the Gem5 simulation as the number of processing cores is varied from 4 to 16. Let T_M be the model training time. In Lee's method, N_{Extra} is the number of training data points used to train the contention model and penalty model, $T_{M_{Extra}}$is the model training time for contention model and penalty model. Similarly, in our method, N_{Mesh} is the number of training data points used to train interconnect model, $T_{M_{Mesh}}$is the model training time for interconnect model. We compare our proposed hierarchical modeling framework to a naïve simulation based method and a hierarchical contention based approach proposed by Lee et al.

Equation 3.7 models the training cost of the naïve simulation based method where the total training time cost is the simulation time plus the modeling time. Equation 3.8 models the training cost of Lee’s contention based method. The lower bound is the best time cost which assumes that the clusters are homogeneous, and only one cluster model needs to be constructed. The upper bound is the worst time cost where the clusters are heterogeneous and the model of each cluster needs to be constructed separately. In Lee's method, the processing core modeling time and contention modeling time is similar. The time cost of our hierarchical modeling framework is described in Equation 3.9. Here we can neglect the interconnect modeling time since it is small compared to the cluster modeling time. Table 3.10 shows the speed up for our proposed hierarchical many-core modeling methodology and Lee’s contention based methodology as compared to the naïve simulation based method. The naïve simulation is $O(n^r)$ with the number of clusters while our proposed method is $O(n)$ at the worst case when the clusters are heterogeneous, and $O(1)$ when the clusters are homogeneous.

$$T_{total} = N * T_C * n^r + T_M \tag{3.7}$$

$$\begin{cases} T_{total}^{upper} = N * T_C * n + N_{extra} * T_C * n^r + n * T_M + T_{M_{extra}} \\ \approx N * T_C * n + N_{extra} * T_C * n^r + (n+2) * T_M \\ T_{total}^{lower} = N * T_C + N_{extra} * T_C * n^r + T_M + T_{M_{extra}} \\ \approx N * T_C + N_{extra} * T_C * n^r + 2 * T_M \end{cases} \tag{3.8}$$

$$\begin{cases} T_{total}^{upper} = N * T_C * n + n * T_M + N_{Mesh} * T_{Mesh} + T_{M_{Mesh}} \\ \approx N * T_C * n + n * T_M \\ T_{total}^{lower} = N * T_C + T_M \end{cases} \tag{3.9}$$

Table 3.10: Speedup comparison of three different modeling methods, normalized to the naïve entire processor simulation with modeling

Method	**Lower bound**	**Upper bound**
Naïve	1	1
Lee's	$\frac{N * n^r}{N + N_{extra} * n^r}$	$\frac{N * n^r}{N * n + N_{extra} * n^r}$
Ours	n^r	n^{r-1}

3.5 Related Work

In this section, we only briefly review the literature with emphasizes on hierarchical design framework.

Lee et al. [1] minimize many-core simulation times in estimating performance through composable regression models for baseline uniprocessor performance, cache contention, and delay penalty. Their uni-core 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. A uniprocessor spline regression model predicts the baseline performance of each core while a contention spline 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. However, in their case, the contention model is still developed cycle-accurately. But the number of data samples collected for contention model is far less than the uniprocessor model. A median CPI error of 6.6% is reported for quad-core processors. A major disadvantage of their work is the lack of scalability of the methodology due the necessity of having to perform detailed simulations of a many-core processor to develop the contention model. The authors have only focused on developing regression models for predicting CPI and not for power estimation.

Datta [51] presents a design exploration framework which employs an instruction trace-driven cycle-accurate simulator to accurately measure power and performance of embedded heterogeneous many-core processors based on a network IP packet processing embedded application. He employs a hierarchical modeling and exploration framework. In the first step, the cores are simulated cycle-accurately with core-level parameters, and the power and performance of the cores are modeled using linear regression. The cores are then optimized using Genetic Algorithm to generate a set of best optimized core microarchitectures with minimal power dissipation. In the last step, simulated annealing is used to optimize the core and memory simultaneously. In this step, only the core to L2 cache and L2 cache to memory packets are simulated with mem-level parameters using a coarse grained simulation. Unfortunately, such a coarse grained simulation is not sufficient to capture the dynamics of complex global interconnects of many-core processors. In our work, we use a detailed cycle accurate simulation to generate accurate global interconnect models. Also, in Datta’s exploration framework, only one optimal micro-architectural design point could be obtained, while in our framework, a Pareto optimal performance-power front is generated.

Cassidy and Andreou [59] establish a system-level analytical model for parallel computational architectures which focuses on first-order system-level effects by extending the Amdahl’s law. They model the CPI and energy costs based on the benchmark parallelization percentage, the area of the processor core and L2 cache, and other fixed area sizes. The many-core processor exploration is then regarded as minimizing the energy-delay product under the constraint of total die area. The inaccuracy of analytical modeling lies in the very coarse level abstraction of the processors, involving only system-level parameters. On the other hand, the statistical machine learning models used in our work is very detailed with a large number of micro-architectural internal parameters.

3.6 Conclusions

Traditional technique of modeling the performance of processors through detailed simulations is not scalable to many-core processor incorporating tens to hundreds of processor cores due to the prohibitive simulation times. For design space exploration of such a many-core architecture thousands of such design points will have to be simulated to determine designs that meet performance, power, and area constraints. Use of statistical performance models serves as a computationally efficient substitute to determining processor performance. Unfortunately, even generating the training data set required for modeling the many-core processor requires detailed cycle accurate simulation of hundreds of micro-architectural configurations with a power law increase in the simulation time with the number of cores.

In this chapter, we have presented a hierarchical modeling methodology to reduce the simulation time. The methodology combines statistical models of individual processing cores, memory hierarchy, and interconnect through a few shared parameters. The resulting many-core model is a cross-coupled combination of the individual models. Evaluations of our many-core models constructed using the MARS algorithms, shows an R^2 ranging from 0.82 – 0.89 for selected Parsec benchmarks with the simulation time increasing linearly in the worst-case with the number of cores.

Our work can be extended in a number of ways. The injection rate of each cluster could be treated as a separate predictor variable for the global interconnect model. The contention on the network could be captured more accurately, by dividing the individual applications into compute and memory access phases, and separately developing injection rate models for these phases. The heterogeneity of the many-core processor could be extended by considering GPUs and DSPs as part of the clusters.

CHAPTER 4: RUN-TIME CROSS-STACK ENERGY OPTIMIZATION

4.1 Introduction

In this chapter, we present a statistical machine learning (SML) based modeling and exploration framework that can be used to rapidly explore vast design spaces of tens of millions of operating points. A number of powerful SML algorithms have been reported in the literature that can capture complex nonlinearities among the tuning variables. From a training set composed of a small fraction of operational points (< 1%), we construct a SML model with both micro-architectural and application parameters as predictor variables. The model predicts the power and performance of the operating points outside the training set. We construct a multi-adaptive regression spline (MARS) based model that uses a number of architecture and application parameters as predictor variables to predict performance and power. We use a Latin Hypercube based sampling strategy to obtain the training set. We employ a feature reduction algorithm to identify the parameters that most significantly contributes to performance and power, thus reducing the operating space to be explored. Since we seek to optimize multiple objectives in an unbiased fashion, we use a Pareto front exploring evolutionary algorithm that uses the MARS model to determine operating points for optimal power and performance. The operating points constituting the Pareto front can be stored in look-up tables for rapidly determining the optimal operating point.

We apply the proposed framework to an x264 video encoding application obtained from the Parsec benchmark. The target architecture is a quad-core processor with the cores based on the Alpha 21264 out-of-order processor. The cores have private L1 data and instruction cache, and a unified cache coherent L2 cache shared through a bus based interconnect. We simulate the performance using the Gem5 architectural simulator and estimate power using the McPAT modeling framework targeting the 22nm technology node. Note that the architecture is not specifically designed for video processing but rather chosen to illustrate our modeling and exploration framework. The micro-architectural predictor variables include a total of 10 core and cache parameters. The application predictor variables include the video resolution, and visual quality determined by the choice of the motion estimation algorithm. The model outputs the average frames per second (FPS) and the average power consumption. The MARS model has an R^2 of 0.9657 and 0.9467 and RMSE (root mean squared error) of 1.829 and 0.0124 respectively for FPS and power consumption. Comparison of the power consumption of Pareto optimal operating point at a lower visual quality to that of Pareto optimal point at a higher visual quality for an x264 video encoder executing on a prototype quad-core processor indicates a power saving of 55%.

The rest of the chapter is organized as follows - in Section 4.2 we describe our SML modeling and exploration framework in detail. We present the evaluation setup and the evaluation results in Section 4.3 and 4.4 respectively. In Section 4.5, we discuss the feasibility of run-time tuning of micro-architectural and application parameters. In Section 4.6, we review related work on run-time cross-stack energy optimization and in Section 4.7, we conclude the chapter identifying possible extensions of our work.

4.2 Statistical Machine Learning Modeling and Exploration Framework

In this section we describe our statistical machine learning (SML) based regression modeling and exploration framework for jointly modeling the power and performance of the computing stack. We use as predictor variables the micro-architectural and application parameters that we seek to tune at run-time. The exploration algorithm utilizes the SML model to determine the Pareto optimal power-performance fronts. Our SML modeling framework is shown in Figure 4.1. In the following subsection we describe the target architecture, and the different components of the framework.

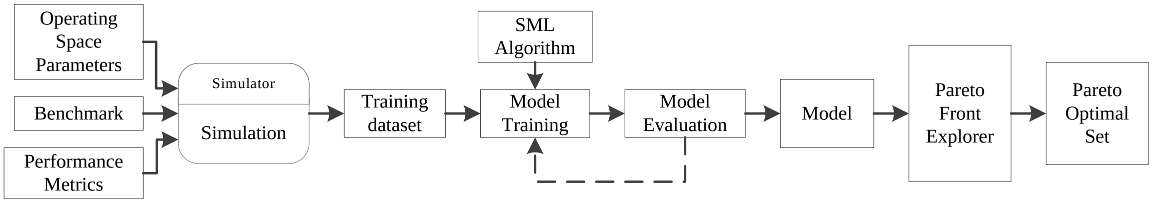

Figure 4.1: Proposed statistical machine learning modeling and exploration framework

4.2.1 Target Architecture and Design Space

The SML algorithm evaluation framework of Figure 4.1 can be applied to any architecture of interest. However, to keep our discussions concrete, in our work we choose to model a quad core multi-core architecture, with each core having a private L1 cache and 4 cores sharing an L2 cache. The individual cores are out-of-order and are based on the Alpha 21264 processor. Each out-of-order core is a single threaded four issue processor with split instruction and data caches. The L1 and L2 caches are non-blocking with miss status holding register (MSHR) and write buffers (WB) for read and write misses. Cache coherency is maintained among the 4 cores sharing the L2 cache. The cache replacement policy is LRU and the cache coherence protocol is bus-based MOESI snooping protocol. Our local memory system is a classic bus-based model. The bus arbitration follows first-come-first-serve logic, and uses round-robin scheduling for bus accesses.

The micro-architectural parameters that we seek to tune are grouped as core and memory parameters. The complete list of parameters that we consider in our work is given in Table 4.1. The possible range of values assumed by the core, memory and network parameters are derived from for commercial processors and from the literature. Note that the total operating space considering the micro-architectural parameters alone is 2.5 million points.

Table 4.1: Micro-architectural parameters and parameter value range considered in our work

Parameters	**Value ranges**
Number of integer Alu-MultDiv	2-1, 4-1, 4-2
Number of floating point Alu-MultDiv	1-1, 2-1, 4-1
Number of physical integer registers	64, 128, 196
Number of physical floating point registers	64, 128, 196
Number of store-load queue entries	32-48, 32-64, 48-64
Number of instruction queue entries	16, 32
Number of reorder buffer entries	128, 160
Number of branch target buffer Entries	2048, 4096
L1 cache size (KB)	32, 64
L1 cache associativity	4, 8
DTLB size	32, 64, 128
ITLB size	32, 64, 128
L2 cache size (MB)	1, 2, 4
L2 cache associativity	8, 16
Core frequency (GHz)	1, 1.5, 2.0
Number of cores	2, 4

We choose power consumption and benchmark specific performance metrics such as task throughput as the output performance metrics. Note that our choice of the architectural parameters, their possible range of values, and the output performance metrics are only illustrative. The proposed framework can be used for any set of predictor variables and performance metrics.

4.2.2 Data Sampling

In order to create a machine learning model, a training subset is generated by sampling the overall design space. The quality of the selected design points is very critical, because all the inferences and predictions of the test design points are based on them. The commonly used sampling methods include random sampling, systematic sampling [29] and stratified sampling [30]. In random sampling the design points are randomly generated. In systematic sampling, the selection of the design points is based on a fixed distance metric between the design points. In stratified sampling the design points are divided into subgroups and the selection of design points from each subgroup is done by either random sampling or systematic sampling. However, none of these methods guarantee an even distributed sampling across the overall design space.

Recently, Latin Hypercube Sampling (LHS) has been proposed by researchers [31] [32] as an alternative sampling method. In LHS, to obtain N design points from a P-dimensional design space, each dimension is divided by N hyperplanes resulting in a total number of N^P small hypercubes. At most one design point is selected from each hypercube. The design point is selected by either Maximum Minimum (*MaxiMin*) Distance LHS criteria or Reduce Correlation (*Correlation*) Distance LHS criteria. In *MaxiMin*, the minimum distance between the selected design points is maximized. The distance can be calculated using a distance metric, such as the Euclidean distance [60]. In *Correlation*, the correlation between the selected design points is minimized. These two criteria enable LHS to guarantee a relatively uniformly distributed design points to be sampled from the design space. In our work, we choose the *MaxiMin* Distance LHS criteria implemented by a Matlab function: *lhsdesign* [33]. The output of *lhsdesign* is a normalized $N \times P$ matrix. We denormalize the entries in this matrix to the possible values in the range of each predictor variable.

4.2.3 Feature Reduction

Feature reduction is a procedure to identify the significance of each predictor variable, and select a subset of most relevant predictor variables to build the learning models for given target variables. To select the optimal subset of P predictor variables, a total number of $P!$ possible subsets need to be evaluated. Such an exhaustive search is only feasible for a small P. Heuristic methods that search through a reduced number of subsets are commonly used for variable selection. Examples of the heuristic methods for variable selection are stepwise forward selection/backward elimination methods [39], genetic algorithms [40] and simulated annealing [41]. In our work, we choose the *rfe* function provided in *caret* R package [43] which uses the stepwise backward elimination with *k*-fold cross validation. In stepwise backward elimination, the procedure starts with the full set of predictor variables as the current subset. In each iteration, the model with current subset of predictor variables is first evaluated using a model independent metric such as the *t*-statistic or R^2 values. Then each predictor variable is eliminated from the current subset individually and the model is re-evaluated using the same metric. The predictor variable which leads to the smallest change of the metric is considered the least significant predictor variable in this iteration and eliminated from current subset for the next iteration. The elimination process stops when a threshold on the model accuracy has been reached, or all the predictor variables have been eliminated from the current subset. The optimal subset of predictor variables will be the one with the most accurate model.

In *k*-fold cross validation, the training data is first partitioned into equal-sized *k* folds. The model training process is then performed in successive *k* rounds. In each round, a different fold is selected for model evaluation using the metrics mentioned above and the other *k*-1 folds are used for model training. This results in *k* lists of significance of each predictor variable in each iteration of the elimination process. The least significant predictor variable is selected based on an average of the significance in the *k* lists.

4.2.4 SML Modeling: Multivariate Adaptive Regression Splines

Linear regression is the most widely used model to construct the approximation function $\hat{f}$ for each target variable using a linear relationship:

$$y = \hat{f}(\boldsymbol{x}) + \epsilon = b + \sum_{i=1}^{P} \omega_i x_i + \epsilon \tag{4.1}$$

Here y is the target variable of interest, b is an intercept term, ϵ is an error term, and ω_i is the corresponding coefficient of predictor variable x_i. A regression spline extends linear regression by using two or more polynomial fittings in each of the $K+1$ disjoint regions separated by K points (knots). Multivariate Adaptive Regression Splines builds on regression splines through the use of truncated power basis functions [14] and adaptive knot selection strategy [10] to construct the approximation function $\hat{f}$:

$$\hat{f}(\boldsymbol{x}) = \sum_{m=0}^{M} \omega_m B_m(\boldsymbol{x}) \tag{4.2}$$

Here M is the number of basis functions included in the model. One representation of these basis functions is:

$$B_m(\boldsymbol{x}) = \prod_{i=1}^{I_m} \left[s_{(i,m)} \left(x_{j_{(i,m)}} - t_{(i,m)} \right) \right]_+^Q \tag{4.3}$$

Here I_m is the number of factors (interaction order) of the m^{th} basis function and (i, m) denotes the m^{th} basis function and the i^{th} interaction order. $s_{(i,m)} = \pm 1$ indicates the positive (right) or negative (left) of the truncated power function pairs of the mth basis function. $x_{j_{(i,m)}}$ is one of the predictor variable $\{x_j\}_{j=1}^{P}$ and $t_{(i,m)}$ is the knot location.

The MARS algorithm employs a two-step procedure to build the model: the forward pass and backward pass. In the forward pass, the algorithm starts with one basis function which is just an intercept term, and then repeatedly adds new basis functions in pairs (two at a time) to the model. In the $(M + 1)^{th}$ iteration, the basis function which maximizes the reduction in SSE (sum of squared error) of the model built in the previous M iterations is selected, and here SSE is described by

$$SSE(M+1) = \sum_{i=1}^{N} \left[y_i - \hat{f}_M(\boldsymbol{x}_i) - B_{(2M+1)}(\boldsymbol{x}_i) - B_{(2M+2)}(\boldsymbol{x}_i)\right]^2 \qquad (4.4)$$

In Equation 4.4, all possible choices of the newly added basis function $B_{(2M+1)}(\boldsymbol{x}_i)$ and $B_{(2M+2)}(\boldsymbol{x}_i)$ are evaluated until the change of SSE of the model is below a given threshold or until the maximum number of basis functions is reached. At the end of the forward pass, the number of basis function is typically chosen to be substantially larger than would be optimal, and results in an over-fitted model. In the backward pass, the MARS algorithm prunes the model built in the forward pass by removing the least effective basis functions one at a time until only the intercept term is left. The least effective basis function is determined based on a modification of the Generalized Cross Validation (GCV) criterion [15] as described by:

$$GCV(M) = \frac{1}{N} \sum_{i=1}^{N} \frac{\left[y_i - \hat{f}_M(\boldsymbol{x}_i)\right]^2}{\left(1 - \frac{enp(M)}{N}\right)^2} \qquad (4.5)$$

$$enp(M) = M + c * \frac{M-1}{2} \qquad (4.6)$$

Here M is the number of basis functions in the model $\hat{f}_M(\boldsymbol{x})$ (including the intercept term), enp is the effective number of parameters given by Equation 4.6, c is the GCV penalty per knot and $(M-1)/2$ is the number of knots. The final output model is the one with the lowest GCV value. The numerator of Equation 4.5 is the MSE on the training data and the denominator represents a penalty for increasing model complexity (number of knots).

In our work, we use the implementation in the *caret* R package [43].

4.2.5 Pareto Front Exploring Evolutionary Algorithm

A multi-objective optimization problem aims to find a set of optimal solutions for multiple objectives simultaneously subject to certain constraints. In general, these multiple objectives may conflict with each other. The optimal solutions are known as Pareto optimal (non-dominated) solutions, which constitutes a Pareto front. Pareto optimal solution describes a situation where no further optimization of any of the objectives is possible without sacrificing the other objectives.

Evolutionary algorithms (EAs) are popular approaches to solve the multi-objective optimization problems. EAs are metaheuristic computational methods, which improve candidate solutions to the optimization problem iteratively given specified solution quality measurements. Among EAs, genetic algorithm (GA) is one of the most popular types. GA is an iterative algorithm which improves the population (solutions) generation by generation through crossover and mutation operations. In our work, we use the Non-dominated Sorting Genetic Algorithms-II (NSGA-II) algorithm [45] to determine the Pareto front. The algorithm is based on Pareto-ranking schemes, in which the population is ranked and the fitness value of each individual is assigned according to the domination relationship and density information (crowding distance) [61]. The dominance rule is defined as follows: consider individuals $\boldsymbol{x}$ and $\boldsymbol{y}$ with corresponding L multi-objectives $\{f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_L(\boldsymbol{x})\}$ and $\{f_1(\boldsymbol{y}), f_2(\boldsymbol{y}), \dots, f_L(\boldsymbol{y})\}$. If for any $i \in (1, \dots, L)$, there exists $f_i(\boldsymbol{x}) < f_i(\boldsymbol{y})$, then $\boldsymbol{x}$ is dominating $\boldsymbol{y}$, $\boldsymbol{x}$ is non-dominated and $\boldsymbol{y}$ is dominated. In NSGA-II, the crowding distance is the average distance of its two neighboring solutions. During the selection in NSGA-II, the boundary points and lower rank (non-dominated) solutions are selected, and if two solutions have the same rank, the less crowded solution will be selected.

We use the NGPM (NSGA-II library in Matlab) tool [44] which is implemented using the NSGA-II population selection scheme as described above. NGPM supports integer-only variables required for the micro-architectural parameters and parallel computes of the objective evaluation function.

4.3 Evaluation Methodology

We evaluate the utility of our proposed modeling framework in cross-stack energy optimization through a video encoding benchmark. The performance metrics are power and frames encoded per second (FPS). The predictor variables include micro-architectural and the application performance tuning parameters. The training data set for constructing the MARS model are generated through detailed simulation of the execution of the benchmark on the target architecture. The simulation tools used and the x264 benchmark are described in this section.

4.3.1 x264 Video Encoding Benchmark

We select x264, an H.264/AVC (Advanced Video Coding) video encoder available in the Parsec [3] benchmark set as our target application to demonstrate our modeling framework. x264 has been widely used in applications ranging from video conferencing to movie distribution, on personal computers, mobile phones and tablets. In PARSEC, x264 is parallelized using the pipeline model, in which each thread encodes an input video frame. The number of frames that can be processed simultaneously is equal to the number of encoder threads that can be executed in parallel.

Among the various qualities of services (QoS) for video, we select the two most intuitive metrics: resolution and visual quality. The video resolution directly determines the encoding performance (FPS) and power consumption of a sequence of video frame sequences. With a larger resolution, for the same compute capability, the FPS is lower and/or the power consumed is higher. We select Motion Estimation (ME) method as the visual quality metric. In typical videos, adjacent video frames are similar and changes are due to the movement of the objects in the frames. Video encoder takes advantage of this characteristic by dividing the frame into a number of blocks with small number of pixels, such as 16x16 or 8x8, and only encodes the motion vector of each block from current frame to the reference frame during video compression. By using this technique, video encoding reduces the computations needed to encode all the original pixels in each frame while requiring fewer encoding bits. To find the motion vector of each block, conventional block matching algorithms for motion estimation such as diamond (DIA), and uneven multi-hexagon (UMH) searching are used. These searching strategies are all composed of multiple-step searching and mainly differ on the search pattern, which is the number of adjacent blocks considered for each step, and the step size of the adjacent blocks. To search for the block B_r in reference frame R corresponding to the block B_c in current frame C, initially, the block of the same position of B_c in R is regarded as the center for searching. Then the differences between B_c with the adjacent blocks are calculated. The best matching adjacent block is the block with the least difference, and this becomes the center for the next step. The differences can be calculated using either Mean Absolute Difference (MAD) or Mean Square Error (MSE). For a $N \times N$ block, MSE and MAD are described as follows:

$$MSE(\Delta i, \Delta j, \Delta t) = \frac{1}{N^2} \sum_{i=0}^{N} \sum_{j=0}^{N} [g(i + \Delta i, j + \Delta j, t + \Delta t) - g(i, j, t)]^2 \qquad (4.7)$$

$$MAD(\Delta i, \Delta j, \Delta t) = \frac{1}{N^2} \sum_{i=0}^{N} \sum_{j=0}^{N} |g(i + \Delta i, j + \Delta j, t + \Delta t) - g(i, j, t)| \qquad (4.8)$$

Here Δi and Δj are block displacements, $(\Delta i, \Delta j, \Delta t)$ determines motion vector of current block in the reference frame which has a Δt difference in time from current frame. This process continues until the adjacent blocks are not any better than the center or when the maximum number of search steps is reached. The motion vector from the initial block to the final center is thus obtained. DIA is the simplest searching strategy providing the fastest speed for motion estimation. In DIA, the two up, two left, two down and two right blocks are checked in each step. UMH is the most common strategy of motion estimation and utilizes unsymmetrical-cross, large hexagon, uneven multi-hexagon-grid, and small hexagon in each step to obtain the best matching block. As a result, a power-performance trade off exists between the two motion estimation algorithms, with visual quality being better for the slower running UMH as compared to the DIA algorithm.

In our work, the choice of DIA and UMH algorithms for motion estimation is considered as two possible operating points of the video encoding application.

4.3.2 Simulation Tools

4.3.2.1 Processor performance modeling

We use the Gem5 simulator [26] for the performance modeling of our target system. Gem5 is an event-driven cycle-accurate simulator, which provides a highly configurable simulation framework, multiple ISAs, diverse CPU models, multiple cache coherence protocols and interconnects models. It also supports both full-system and system-call emulation modes. The full-system simulation mode simulates a complete computer system including operating system kernel and I/O devices, while in system-call emulation mode, the common system calls are emulated by calling the host OS. Besides, it provides a clear interface for check-pointing, fast-forwarding, debugging and statistics.

Our Alpha ISA based architecture described in Section 4.1.1 is simulated in full-system mode, which models a complete Linux system with kernel version 2.6.27. We run the x264 benchmark on Gem5 by fast forwarding the simulation starting point to the pre-defined start of region of interest (ROI), which is the parallel execution phase. We start collecting the statistics such as the number of integer/floating point instructions, the number of accesses and misses to each hardware unit and the total number of cycles when the benchmark is running in the ROI and finish when the benchmark leaves the ROI. We use the input videos provided in PARSEC and use FFMPEG [62] to change the video resolutions.

4.3.2.2 Processor power modeling

We use McPAT 0.8 [28] for the power modeling of our target architecture. McPAT is an integrated power, area, and timing hierarchical modeling framework for multi-core and many-core processor configurations targeting technologies ranging from 90nm to 22nm. McPAT supports both static and dynamic power modeling at the micro-architectural level, circuit and technology level.

At the micro-architectural level, McPAT includes models of major architectural components such as cores, interconnection networks, caches, memory controllers, and clocking. At the circuit level, the architectural building blocks are mapped into four basic circuit structures: hierarchical wires, arrays, complex logic, and clocking networks. At the technology level, the physical parameters of devices and wires, such as unit resistance, capacitance, and current densities, is calculated based on the data from the ITRS roadmap.

4.4 Results

In this section we present results on the accuracy of the MARS models, the Pareto optimal fronts, and the success of the cross-stack energy optimization goals. A training dataset of 360 points is obtained through Latin Hypercube Sampling (see Section 4.2.2) and cycle-accurate simulation of the region of interest of the x264 benchmark (see Section 4.3.2) on an Intel Xeon machine. Each simulation point takes 3 to 7 hours depending on the video frame resolution.

4.4.1 Evaluating the MARS Model Accuracy

In this section, our goal is to show the impact of feature reduction in improving the model accuracy of MARS. First, we identify the most significant micro-architectural parameters using the cross-validation and backward feature reduction as described in section 4.2.3. We then compare the accuracy of the model trained with the selected significant features to the model trained with the full feature set.

In the *caret* R package, the feature reduction procedure is combined with the MARS modeling. Here, both the micro-architectural and x264 application parameters are considered. Because the options for each application parameter are categorical values, we consider each option as a separate variable by assigning 0 or 1 if the option is not taken or taken. We also preprocess the raw training dataset by normalizing all the model input variables (architectural design parameters and application parameter) to 0~1 using the maxi-min normalization as described below:

$$norm(x) = (x - \min(x))/(\max(x) - \min(x)) \tag{4.9}$$

The models are generated in less than 100 seconds on an 8 core Intel Xeon workstation.

Table 4.2 shows the significant micro-architectural operating parameters in FPS modeling and power modeling. Out of the 19 micro-architectural operating parameters of Table 4.1, 7 are selected for FPS modeling and 10 are selected for power modeling. To show the impact of feature reduction on the modeling accuracy, the performance of the MARS model with all features of Table 4.1 is compared against the MARS model with the reduced feature set. As can be seen in Table 4.3 the feature reduction improves the accuracy of the model and reduces the overall design space from 15 million to 0.05 million operating points.

To evaluate the performance of MARS models in predicting the performance of operating points outside the training set, we sampled 20 new test points through LHS

Table 4.2: Significant predictor variables selected after feature reduction for FPS and Power modeling. 9 and 12 out of 21 predictor variables are chosen for FPS and Power modeling respectively

Performance Metrics	**Predictor variables**
FPS (9/21)	Resolution, Motion estimation algorithm, Number of cores, Number of integer ALU, Number of physical integer registers, Instruction queue size, Core frequency, L1 cache associativity, L2 cache size
Power (12/21)	Number of cores, L1 cache size, L2 cache size, Resolution, L1 cache associativity, Core frequency, Motion estimation algorithm, L2 cache associativity, Number of physical integer registers, Number of physical floating point registers, DTLB size, Store queue entries

Table 4.3: R and RMSE comparison between the models with and without feature reduction. The total operating points reduces from 15 million to 0.05 million after feature reduction

Performance Metrics	**FPS modeling**		**Power modeling**	
	With feature reduction (9)	Without feature reduction (21)	With feature selection (12)	Without feature selection (21)
R^2	0.9657	0.9084	0.9467	0.8954
RMSE	1.829	2.566	0.0124	0.1358

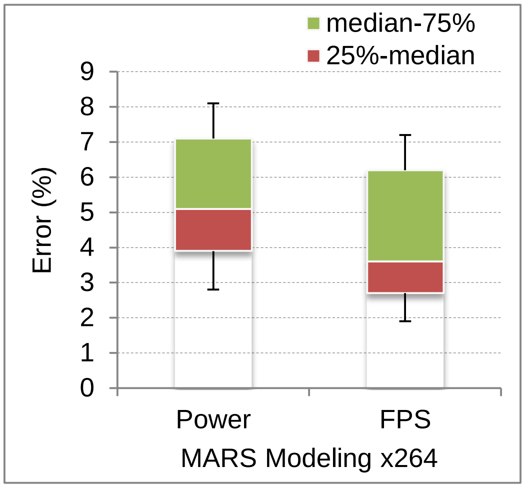

Figure 4.2: Boxplot of the prediction error distribution

sampling strategy with different architectural and application parameter values. Figure 4.2 shows the boxplot of the error percentage distribution. The error is calculated using the values of the FPS and power consumption predicted by the MARS model and that obtained by directly simulating the test points.

4.4.2 Pareto Front: FPS, Power

Figures 4.3 and 4.4 respectively show the power-FPS Pareto front for different video frame resolutions using the DIA and UMH motion estimation method. As expected, a larger FPS is possible at lower video frame resolutions. Also for the same resolution and

FPS, the UMH motion estimation algorithm consumes more power than the DIA algorithm.

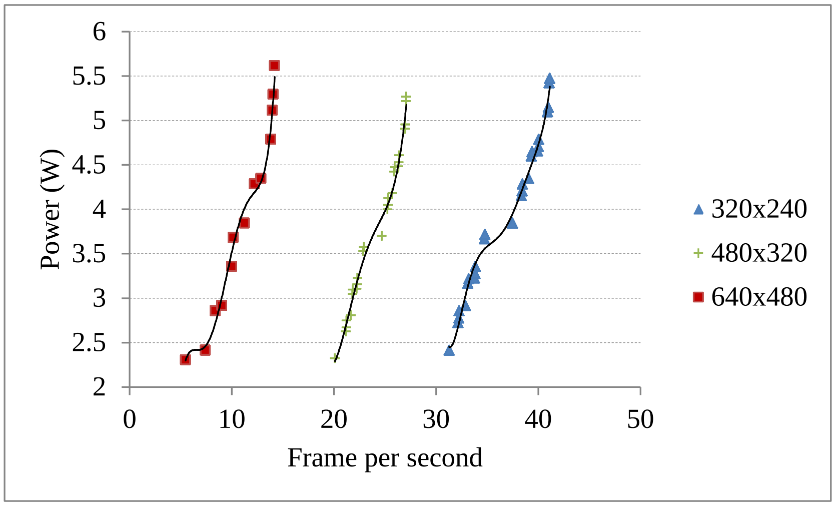

Figure 4.3: Pareto front for different video frame resolutions with motion estimation algorithm DIA

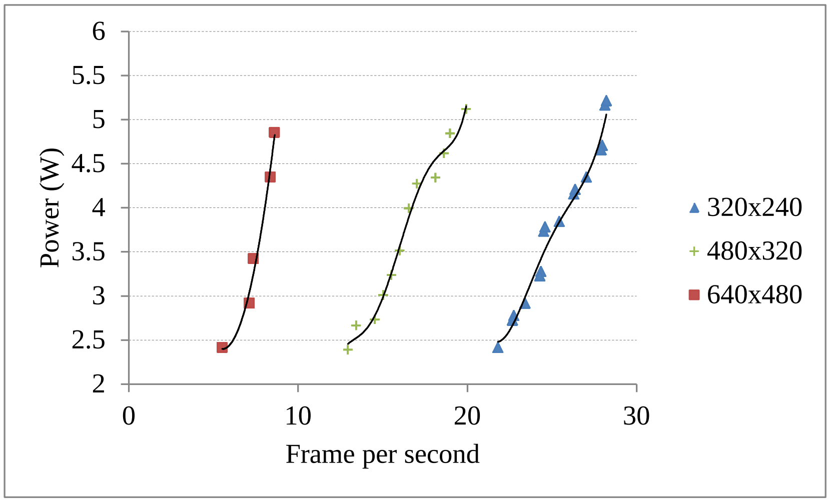

Figure 4.4: Pareto front for different video frame resolutions with motion estimation algorithm UMH

4.4.3 Cross-stack Adaptation Impact

To evaluate our cross-stack adaptation framework for power and energy savings, for a user specified video frame resolution and FPS, we compare the power consumption of Pareto optimal operating point at a lower visual quality to that of Pareto optimal point at a higher visual quality. Here, we assume that the user requires a specified resolution and FPS, but is flexible with the visual quality. For our experiments, we fix the resolution at 480x320 and FPS at 20. Table 4.4 lists the micro-architectural configuration and the corresponding power consumption for the Pareto optimal point for both higher and lower visual qualities. A 55% power-saving is achieved by jointly tuning the micro-architectural configurations and the visual quality. Table 4.4 also shows that when tuning from the Pareto optimal configuration with higher visual quality to the Pareto optimal configuration with lower visual quality, not all the parameters need to be changed. The visual quality is reduced by switching from the UMH to the DIA motion estimation algorithm. The core

Table 4.4: Power consumption and micro-architectural parameter values for the Pareto optimal points at higher and lower visual quality for video frame resolution of 480x320 and FPS of 20

	UMH	**DIA**
Power (W)	5.12	2.32
Number of integer ALU	4	4
Core frequency (GHz)	1.5	1
Number of physical integer registers	128	128
Number of physical floating point registers	64	64
Store queue entries	32	32
Instruction queue entries	32	32
L1 cache size (KB)	64	32
L1 cache associativity	4	4
DTLB size	128	128
L2 cache size (MB)	1	1
L2 cache associativity	8	8
Number of cores	4	2

frequency, L1 cache size are decreased. The number of cores utilized drops from 4 to 2. The Pareto optimal points calculated can be stored in look-up tables. At run-time, for a new operating condition (such as power or FPS requirements), the determination of the operational operating point can thus be done rapidly. Table 4.5 shows the Pareto optimal operating points with highest and lowest FPS and corresponding power consumption for different video frame resolutions and motion estimation algorithms. Note that the total size of Pareto optimal set consists of 60 points.

Table 4.5: Pareto optimal operating points for highest and lowest FPS and corresponding power consumption for different video frame resolutions and motion estimation algorithms

ME algorithms		**DIA**				**UMH**			
Performance Metrics		**FPS**		**Power (W)**		**FPS**		**Power (W)**	
		High	Low	High	Low	High	Low	High	Low
Resolutions	320x240	41	31	5.42	2.41	29	23	4.65	2.77
	480x320	27	20	5.26	2.32	20	12	5.12	2.38
	640x480	15	6	4.74	2.3	9	5	4.34	2.41

4.4.4 Discussion: Ease of Reconfiguration

In this section we briefly review the existing literatures for the feasibility of run-time reconfiguration of micro-architectural and application parameters. In general, the tuning of these micro-architectural parameters on the fly involves either hardware support by adding circuitries to shut down the unused part of each component or to put the unused part into sleep mode. The number of cores, functional units and physical registers can be controlled through clock-gating [63] – [68]. Clock-gating is a method to prevent the clock signal from reaching the selected components by adding an enable condition signal to the components. Clock-gating has a small overhead in terms of the added circuit and a performance overhead of one to few clock cycles [69]. The core operating frequency can be changed through DVS (dynamic voltage scaling) by adjusting the core voltage at run-time [70] [71]. The voltage transition is often achieved through inductor-based voltage regulator. Depending on the technologies used, the voltage transition times are in the order of tens of microseconds [72]. The store, load and instruction queues are all CAM-like (content address memory) structures which can be banked at design time. With these structure implementations, clock-gating technology can be applied to each bank to dynamically tune the queue size [73] [74]. The ROB, BTB and TLB are arrays of CAM which can be partitioned into banks as well and each bank can be disabled separately [75] – [78]. Shutting down these CAM and CAM-like structures helps mitigate leakage power. The bank implementation can also be applied to caches with the cache changed by shutting down the cache banks using clock-gating techniques. The cache associativity can be tuned by logically concatenating neighboring ways [79] – [82]. The cache way concatenation is implemented by adding small and simple cache way select registers. The associativity sets with corresponding partition are then enabled using a mask register. The cache way concatenation and partitioning only require an OS context-switch time and on order of 10~30 cycles [83]. For the application level qualities adaptation, choice of the motion estimation algorithm and the video frame resolutions can be set as global variables in the encoder. The encoder the checks these global variables before encoding each frame [84]. To lower the associated overhead, the check of the global variables need only be done when a triggering event, such as a change in the number of tasks, occurs. In case the application source code is not available, application parameter tuning can be done through binary instrumentation tools (such as the Pin tools [85] for x86) to modify the compiled binaries.

4.5 Related Work

In this section we briefly review the literature on micro-architectural and/or software run-time energy adaptation with an emphasis on previous research where cross-stack energy optimization was used. Regarding run-time adaptation of micro architectural parameters, tuning of frequency [86], cache parameters [79] [83], interconnect parameters [87] have been reported. For application layer run-time energy adaptation [88] [89], the benchmarks were drawn from the multimedia. In [88], tuning parameters studied include the dithering mode of video player, number of hidden layers in a neural network based speech recognizer, sampling rate of audio devices (voice-over-IP). In [89], video bit, video distortion, and scene activity were considered.

Work related to run-time cross-stack energy optimization is presented in [90] [84] [91]. In [90], for a video encoder, the number of cores, the encoder parallelization schemes generated using the MPSoC parallelization assist (MPA) tool, and the core frequency were considered as run-time parameters. At design time, NSGA-II was used to generate the Pareto-set, and each operating point was evaluated using TLMSim. At run-time, an online MMKP (Multidimensional Multiple-choice Knapsack Problem) heuristic strategy was used to select the Pareto optimal from the look-up tables for different tasks depending on FPS requirements. In [84], for video encoding and decoding, CPU frequency, encoder/decoder qualities were chosen as the run-time parameters. At design time, analytical models were derived for power, application cycles, and the allocated task processing time. The power model and application cycles model were respectively obtained via measurements on a laptop by varying the CPU frequency settings and encoder/decoder video qualities. The allocated processing time for each task was calculated using the application cycles and core frequency. At run-time, a global and an internal adaptation scheme were employed. The global adaptation was triggered when a new task was generated or finished. The task utilization was calculated from the task processing time and the task release period was monitored at run-time. The core frequency and video quality were gradually increased or decreased until the maximum task utilization was reached or the CPU utilization was 100%. The internal adaptation was triggered when any task CPU utilization was below a certain threshold by adapting the CPU allocation and/or CPU frequency. In [91], for video applications, cache size, associativity, video qualities and DVS were considered as run-time parameters. At design time, Wattch and SimpleScalar were used to exhaustively simulate all the configurations (cache and DVS) for all the video qualities. For each video quality, the best configuration was chosen as that with the minimum energy consumption. Run-time adaptation was triggered when a new task was created or a current task was finished, and the operating point was selected from the look-up table. For a new task, the quality of the new task was gradually increased from the minimum video quality. At the end of current task, the lowest quality of the remaining tasks was gradually increased.

Compared to these prior works on cross-stack optimization, our proposed modeling and exploration framework can handle a large number of operating parameters and associated operating space consisting of millions of points through the use of machine learning algorithms to efficiently generate Pareto optimal power-performance sets for run-time adaptation.

4.6 Conclusions

In this chapter, we have presented a statistical machine learning based modeling and exploration framework that enables cross-stack optimization of energy at run-time. As the number of micro-architectural, system software, and application and parameters that need to be tuned at run-time grows, a scalable method is required to determine the optimal operating point under dynamically changing operating conditions. We constructed a highly accurate ($R^2 \sim 0.95$) multi adaptive regression spline model from a training data set derived from < 1% of the total operating space. The model uses 10 micro-architectural and 2 application parameters to predict performance and power. We then used the model to derive the power-performance Pareto optimal front which can then be stored in look-up tables for run-time use. In a use-case scenario for extending the battery life, the system operating points are adapted based on the energy availability for the task.

Comparison of the power consumption of Pareto optimal operating point at a lower visual quality to that of Pareto optimal point at a higher visual quality for an x264 video encoder executing on a prototype quad core processor indicates a power saving of 55%. Here the video resolution and throughput were fixed while the visual quality and the micro-architectural parameters were simultaneously tuned. Of the 10 micro-architectural parameters considered only 3 operating parameters (L1 cache size, number of cores, and core frequency) had to be tuned. Similar Pareto optimal tradeoffs between power and FPS are possible for other video resolutions and visual qualities.

Several extensions of the work are possible – our proposed framework can be applied for run-time optimization of other metrics such as temperature and system noise. The micro-architectural parameters that are candidates for tuning can be extended to include interconnection network and non-processor related system parameters such as display settings. In this work we have only considered tuning of the micro-architectural and application level parameters. Future work could extend the computing stack parameters that are tuned to include operating system parameters such as core allocation and scheduling budgets. Also, the statistical machine learning models could be extended to incorporate system events as predictor variables so as to capture program execution dynamics. The models are then evaluated at run-time based on system events captured by hardware event counters. However, in this case, rather than pre-computed look up tables, the Pareto optimal front exploration would also have to be done at run-time.

CHAPTER 5: CONCLUSIONS

In this dissertation, we sought to address the challenge of designing complex many-core processors by establishing a machine learning based modeling frame work. The machine learning models are able to capture the dependence of the performance, power and area of the processor to both micro-architectural and application level parameters to a high degree of accuracy. Training such models requires relatively a small simulation time, since typically only a fraction of the design space (< 0.1%) has to be simulated.

5.1 Summary of Results

In this dissertation, we investigated the suitability of a number of machine learning algorithm in micro-architectural modeling and concluded that the Multivariate Adaptive Regression Splines shows good performance both in terms of accuracy and model construction time. Despite the vast reduction in design time possible with the statistical machine learning (SML) model driven micro-architectural exploration its scalability as the number of cores increase is limited. To achieve our goal of building scalable many-core models with good accuracy, we employed a “divide and conquer” strategy in constructing a hierarchical performance, area, and power models for many-core processors. The entire micro-architectural design space of a many-core processor was divided into cluster-level parameters and interconnect-level parameters. Performance, power, and area models were constructed separately for the clusters and the global interconnect. The interactions between the clusters were captured through two parameters – the interconnect latency, and the average injection rate of packets into the network. In spite of the simplifications introduced by our hierarchical modeling framework, we obtained a good accuracy with this approach with a vastly reduced modeling time. Also, our approach showed a worst case linear scaling with the number of cores as opposed a super-linear power law type scaling observed with previously proposed approaches.

We then applied the statistical machine learning based modeling and exploration framework to enables cross-stack optimization of energy at run-time. As the number of micro-architectural, system software, and application and parameters that need to be tuned at run-time grows, a scalable method is required to determine the optimal operating point under dynamically changing operating conditions. We constructed a highly accurate Multivariate Adaptive Regression Splines model from a training data set derived from < 1% of the total operating space. The model uses 10 micro-architectural and 2 application parameters to predict performance and power. We then used the model to derive the power-performance Pareto optimal front which can then be stored in look-up tables for run-time use. In a use-case scenario for extending the battery life, the system operating points are adapted based on the energy availability for the task.

5.2 Future Work

Although we have compared the performance of a number of machine learning algorithms, a study of other algorithms such as Kriging, Radial Basis Functions can be undertaken. In application areas in mechanical engineering [57] such algorithms have shown comparable performance to Multivariate Adaptive Regression Splines. We can also extend the application to a number of other benchmarks such as SPEC 2006 and EMBC embedded benchmarks. We can also investigate the suitability our model driven approach in the incorporation of higher level architectural parameters such as the type of the instruction set and compiler options.

Our work on developing hierarchical model for many-core processors can be extended in a number of ways. The injection rate of each cluster could be treated as a separate predictor variable for the global interconnect model. The contention on the network could be captured more accurately, by dividing the individual applications into compute and memory access phases, and separately developing injection rate models for these phases. The heterogeneity of the many-core processor could be extended by considering GPUs and DSPs as part of the clusters.

Several extensions of our work on run-time optimization are possible - our proposed framework can be applied for run-time optimization of other metrics such as temperature and system noise. The micro-architectural parameters that are candidates for tuning can be extended to include interconnection network and non-processor related system parameters such as display settings. In this work we have only considered tuning of the micro-architectural and application level parameters. Future work could extend the computing stack parameters that are tuned to include operating system parameters such as core allocation and scheduling budgets. Also, the statistical machine learning models could be extended to incorporate system events as predictor variables so as to capture program execution dynamics. The models are then evaluated at run-time based on system events captured by hardware event counters. However, in this case, rather than pre-computed look up tables, the Pareto optimal front exploration would also have to be done at run-time.

REFERENCES

[1] B. C. Lee, J. Collins, H. Wang and D. Brooks, “CPR: Composable performance regression for scalable multiprocessor models,” in *41th IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 270-281

[2] E. Ipek, S. A. McKee, R. Caruana, B. R. de Supinski and M. Schulz, “Efficiently exploring architectural design spaces via predictive modeling,” in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 195-206

[3] C. Bienia, S. Kuman, J. P. Singh and K. Li, “The PARSEC benchmark suite: characterization and architectural implications,” in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008, ACM, New York, NY, USA, pp. 72-81

[4] P. Baldi and S. Brunak, Bioinformatics: The machine learning approach, MIT Press, 2001

[5] P. Stone and M. Veloso, “Multiagent systems: A survey from a machine learning perspective,” in *Autonomous Robots*, vol. 8, no. 3, pp. 345-383, 2000

[6] H. Chen, “Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms,” in *Journal of the American Society for Information Science,* vol.46, no. 3, pp. 194-216, 1995

[7] L. J. Cao and F. E. H. Tay, “Support vector machine with adaptive parameters in financial time series forecasting,” in *IEEE Transactions on Neural Networks*, vol. 14, no. 6, 2003

[8] C. Zhang, A. Ravindran, K. Datta, A. Mukherjee and B. Joshi, “A machine learning approach to modeling power and performance of chip multiprocessors,” in *29th IEEE International Conference on Computer Design*, 2011, pp. 45-50

[9] M. Brown and C. Harris, *Neurofuzzy Adaptive Modeling and Control*, Hertfordshire, UK: Prentice Hall Int. Ltd., 1994

[10] P. L. Smith, “Curve fitting and modeling with splines using statistical variable selection techniques,” Hampton, NASA, Langley Research Center, VA, Technical Report, nasa_techdoc_19830005637, Nov. 1, 1982

[11] J. H. Friedman and C. B. Roosen, “An introduction to multivariate adaptive regression splines,” in *Statistical Methods in Medical Research*, vol. 4, pp. 197-217, Sep. 1995

[12] R. E. Bellman, *Adaptive control processes: A guided tour*, Princeton, NJ: Princeton Univ. Press, 1961

[13] C. M. Bishop, *Neural networks for pattern recognition*, New York: Oxford Uni. Press, 1996

[14] J. H. Friedman, “Multivariate adaptive regression splines,” in *Annals of Statistics*, vol. 19, no. 1, pp. 1-67, 1991

[15] T. Hastie, R. Tibshirani and J. Friedman, The elements of statistical learning: Data Mining, Inference, and Prediction, New York: Springer, 2003

[16] T. M. Mitchell, *Machine Learning*, New York: McGraw-Hill, 1997

[17] C. Lin, “Optimization, support vector machines, and machine learning,” Talk at DIS, University of Rome and IASI, CNR, 2005

[18] A. Moghaddamnia, M. G. Gousheh, J. Piri, S. Amin and D. Han, “Evaporation estimation using artificial neural networks and adaptive neuro-fuzzy inference system techniques,” in *Advances in Water Resources*, vol. 32, pp. 88-97, Jan. 2009

[19] C. Cortes and V. Vapnik, “Support-vector networks,” in *Machine Learning*, vol. 3, pp. 273-297, Sep. 1995

[20] C. Chang and C. Lin, “LIBSVM: A library for support vector machines,” in *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 27, pp. 1-27, May 2011

[21] C. J. C. Burges, “A tutorial on support vector machines for pattern recognition,” in *Data Mining and Knowledge Discovery*, vol. 2, pp. 121-167, Jun. 1998

[22] M. I. Jordan, 2004, *The kernel trick* [Online]. Available: http://www.cs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf

[23] B. Scholkopf and A. J. Smola, Learning with kernels: Support vector machines, regularization, optimization, and beyond, MIT Press, Cambridge, MA, USA, 2001

[24] F. R. Bach and M. I. Jordan, “Kernel independent component analysis,” in *Journal of Machine Learning Research*, vol. 3, pp. 1-48, Mar. 2003

[25] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis*, New York: Cambridge University Press, 2004

[26] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill and D. A. Wood, “The gem5 simulator,” in *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1-7, 2011

[27] A. R. Alameldeen and D. A. Wood, “IPC considered harmful for multiprocessor workloads,” in *IEEE Micro*, vol. 26, pp. 8-17, Jul.-Aug. 2006

[28] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman,D. M. Tullsen and N. P. Jouppi, “McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 469-480

[29] H. Gundersen and E. Jensen, “The efficiency of systematic sampling in stereology and its prediction,” in *Journal of Microscopy*, vol. 147, pp. 229-263, Sep. 1987

[30] M. D. McKay, R. J. Bechman and W. J. Conover, “A comparison of three methods for selecting values of input variables in the analysis of output from a computer code,” in *Technometrics*, vol. 42, no. 1, pp. 55-61, Feb. 2000

[31] S. Bird, “Software knows Best: A Case for Hardware Transparency and Measurability,” M.S. thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA, 2010

[32] G. Esmeraldo and E. Barros, “A Genetic Programming based approach for efficiently exploring architectural communication design space of MPSoCs,” in *Programmable Logic Conference*, VI Southern, 2010, pp. 29-34

[33] MathWorks Inc., 2012, *Latin hypercube sample* [Online]. Available: http://www.mathworks/com/help/stats/lhsdesign.html

[34] S. Chakrabarti, E. Cox, E. Frank, R. H. Guting, J. Han, X. Jiang, M. Kamber, S. S. Lightstone, T. P. Nadeau, R. E. Neapolitan, D. Pyle, M. Refaat, M. Schneider, T. J. Teorey and I. H. Witten, *Data mining: Know it all*, Burlington, MA: Morgan Kaufmann, 2008

[35] E. M. Knorr and R. T. Ng, “Algorithms for mining distance-based outliers in large datasets,” in *Proceedings of the 24th International Conference on Very Large Data Bases*, 1998, pp. 392-403

[36] Z. He, X. Xu and S. Deng, “Discovering cluster-based local outliers,” in *Pattern Recognition Letters*, vol. 24, pp. 1641-1650, Jun. 2003

[37] V. Hautamaki, S. Cherednichenko, I. Karkkainen, T. Kinnunen and P. Franti, “Improving k-means by outlier removal,” in *Proceedings of the 14th*

Scandinavian Conference on Image Analysis, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 978-987

[38] R Development Core Team (2008). “R: A language and environment for statistical computing,” R Foundation for Statistical Computing, Vienna, Austria

[39] S. Derksen and H. J. Keselman, “Backward, forward and stepwise automated subset selection algorithms: Frequency of obtaining authentic and noise variables,” in *British Journal of Mathematical and Statistical Psychology*, vol. 45, pp. 265-282, Nov. 1992

[40] D. Broadhurst, R. Goodacre, A. Jones, J. J. Rowland and D. B. Kell, “Genetic algorithms as a method for variable selection in multiple linear regression and partial least squares regression, with applications to pyrolysis mass spectrometry,” in *Analytica Chimica Acta*, vol. 348, pp. 71-86, Aug. 1997

[41] J. M. Sutter, S. L. Dixon and P. C. Jurs, “Automated descriptor selection for quantitative structure-activity relationships using Generalized Simulated Annealing,” in *Journal of Chemical Information and Modeling*, vol. 35, pp. 77-84, Jan. 1995

[42] J. Han, M. Kamber and J. Pei, *Data mining: Concepts and techniques*, 3rd ed., San Francisco, CA: Morgan Kaufmann, 2011

[43] M. Kuhn, J. Wing, S. Weston, A. Williams, C. Keefer and A. Engelhardt, “caret: Classification and regression training,” R package version 5.15-044, 2012

[44] S. Lin, 2011, *NGPM – A NSGA-II Program in Matlab* [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/31166

[45] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, “A fast and elitist multi-objective genetic algorithms: NSGA-II,” in *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182-197, Apr. 2002

[46] P. J. Joseph, K. Vaswani and M. J. Thazhuthaveetil, “A predictive performance model for superscalar processors,” in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 161-170

[47] C. Dubach, T. Jones and M. F.P. O’Boyle, “Microarchitectural design space exploration using an architecture-centric approach,” in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 262-271

[48] T. S. Karkhanis and J. E. Smith, “Automated design of application specific superscalar processors: an analytical approach,” in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 2007, pp. 402-411

[49] T. Sherwood, M. Oskin and B. Calder, “Balancing design points with Sherpa,” in Proceedings of the 2004 International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, 2004, pp. 57-68

[50] S. Kang and R. Kumar, “Magellan: a search and machine learning-based framework for fast multi-core design space exploration and optimization,” in *Proceedings of the Conference on Design, Automation, and Test in Europe*, 2008, pp. 1432-1437

[51] K. Datta, “An efficient design space exploration framework to optimize power-efficient heterogeneous many-core multi-threading embedded processor architectures,” Ph.D. dissertation, Dept. of Electrical and Computer Engineering, Univ. of North Carolina at Charlotte, Charlotte, NC, 2011

[52] T. Wolf, “Challenges and applications for network-processor-based programmable routers,” in *2006 IEEE Sarnoff Symposium*, 2006, pp. 1-4

[53] S. Mysore, B. Agrawal, F. T. Chong and T. Sherwood, “Exploring the processor and ISA design for wireless sensor network applications,” in *21st International Conference on VLSI Design*, 2008, pp. 59-64

[54] Y. Lin, Y. Lin, Y. Lai and K. Tseng, “Modeling and analysis of core-centric network processors,” in *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 41, 2008

[55] M. E. Salehi, H. Dorosti and S. M. Fakhraie, “Architecture-level design space exploration of super scalar microarchitecture for network applications,” in *13th Euromicro Conference on Digital System Design: Architectures, Methods, and Tools*, 2010, pp. 269-272

[56] C. Dubach, T. M. Jones and M. F.P. O’Boyle, “Exploring and predicting the architecture/optimizing compiler co-design space,” in *Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2008, pp. 31-40

[57] G. G. Wang and S. Shan, “Review of metamodeling techniques in support of engineering design optimization,” in *Journal of Mechanical Design*, vol. 129, no. 4, pp. 370-380, 2007

[58] N. Agarwal, T. Krishna, L. Peh and N. K. Jha, “GARNET: A detailed on-chip network model inside a full-system simulator,” in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 33-42

[59] A. S. Cassidy and A. G. Andreou, “Beyond Amdahl’s law: An objective function that links multiprocessor performance gains to delay and energy,” in *IEEE Transactions on Computers*, vol. 61, pp. 1110-1126, Aug. 2012

[60] E. W. Weisstein, *Euclidean metric*, [MathWorld—A Wolfram Web Resource]. Available: http://mathworld.wolfram.com/EuclideanMetric.html

[61] A. Konak, D. W. Coit and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," in *Reliability Engineering & System Safety*, vol. 91, pp. 992-1007, Sep. 2006

[62] FFMPEG, 2012. Available: http://www.ffmepg.org

[63] Y. Luo, J. Yu, J. Yang and L. Bhuyan, "Low power network processor design using clock gating," in *Proceedings of the 42nd Annual Design Automation Conference*, 2005, pp. 712-715

[64] J. S. Kim, M. B. Taylor, J. Miller and D. Wentzlaff, "Energy characterization of a tiled architecture processor with on-chip networks," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, 2003, pp. 424-427

[65] S. Kim, S. V. Kosonocky and D. R. Knebel, "Understanding and minimizing ground bounce during mode transition of power gating structures," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, 2003, pp. 22-25

[66] R. Kalla, B. Sinharoy and J. M. Tendler, "IBM Power5 chip: a dual-core multithreaded processor," in *IEEE Micro*, vol. 24, pp. 40-47, Mar.-Apr. 2004

[67] D. Molka, D. Hackenberg, R. Schone and M. S. Muller, "Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system," in *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, 2009, pp. 261-270

[68] R. Maro, Y. Bai and R. I. Bahar, "Dynamically reconfiguring processor resources to reduce power consumption in high-performance processors," in *Proceedings of the First International Workshop on Power-Aware Computer Systems-Revised Papers*, 2000, pp. 97-111

[69] L. Benini, A. Bogliolo and G. D. Micheli, "A survey of design techniques for system-level dynamic power management," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, pp. 299-316, Jun. 2000

[70] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001, pp. 89-102

[71] T. D. Burd and R. W. Brodersen, “Design issues for Dynamic Voltage Scaling,” in *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, 2000, pp. 9-14

[72] W. Kim, M. S. Gupta, G. Wei and D. Brooks, “System level analysis of fast, per-core DVFS using on-chip switching regulators,” in *IEEE 14th International Symposium on High Performance Computer Architecture*, 2008, pp. 123-134

[73] H. Li, Y. Chen, K. Roy and C. Koh, “SAVS: a self-adaptive variable supply-voltage technique for process-tolerant and power-efficient multi-issue superscalar processor design,” in *11th Asia and South Pacific Conference on Design Automation*, 2006, pp. 6

[74] D. Folegnani and A. Gonzalez, “Energy-effective issue logic,” in *Proceedings of the 28th Annual International Symposium on Computer Architecture,* 2001, pp. 230-239

[75] V. Delaluz, M. Kandemir, A. Sivasubramaniam, M. J. Irwin and N. Vijaykrishnan, “Reducing dTLB energy through dynamic resizing,” in *Proceedings of the 21st International Conference on Computer Design*, 2003, pp. 358-363

[76] D. Chaver, L. Pinuel, M. Prieto, F. Tirado and M. C. Huang, “Branch prediction on demand: an energy-efficient solution [microprocessor architecture],” In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, 2003, pp. 390-395

[77] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas and M. L. Scott, “Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling,” in *Proceedings of the 18th International Symposium on High-Performance Computer Architecture*, 2002, pp. 29-40

[78] L. T. Clark, E. J. Hoffman, J. Miller, M. Biyani, L. Liao, S. Strazdus, M. Morrow, K. E. Velarde and M. A. Yarch, “An embedded 32-b microprocessor core for low-power and high-performance applications,” in *IEEE Journal of Solid-State Circuits*, vol. 36, pp. 1599-1608, Nov. 2001

[79] W. Wang, P. Mishra and S. Ranka, “Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems,” in *Proceedings of the 48th Design Automation Conference*, 2011, pp. 948-953

[80] C. Zhang, F. Vahid and R. Lysecky, “A self-tuning cache architecture for embedded systems,” in *ACM Transactions on Embedded Computing Systems*, vol. 3, pp. 407-425, May 2004

[81] G. E. Suh, S. Devadas and L. Rudolph, “Analytical cache models with applications to cache partitioning,” in *Proceedings of the 15th International Conference on Supercomputing*, 2001, pp. 1-12

[82] R. Balasubramonian, D. Albones, A. Buyuktosunoglu and S. Dwarkadas, “Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures,” in *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2000, pp. 245-257

[83] M. Paul and P. Petrov, “Dynamic adaptive I-Cache partitioning for energy-efficient embedded multitasking,” in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, pp. 2067-2080, 2011

[84] W. Yuan, K. Nahrstedt, S. V. Adve, D. L. Jones and R. H. Kravets, “GRACE-1: cross-layer adaptation for multimedia quality and battery energy,” in *IEEE Transactions on Mobile Computing*, vol. 5, pp. 799-815, Jul. 2006

[85] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi and K. Hazelwood, “Pin: building customized program analysis tools with dynamic instrumentation,” in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2005, pp. 190-200

[86] K. Isci, Al. Buyuktosunoglu, C. Cher, P. Bose and M. Martonosi, “An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget,” in *39th Annual IEEE/ACM International Symposium on Microarchitecture*. 2006, pp. 347-358

[87] J. Meng, C. Chen, A. K. Coskun and A. Joshi, “Run-time energy management of manycore systems through reconfigurable interconnects,” in *Proceedings of the 21st edition of the Great Lakes Symposium Great Lakes Symposium VLSI*, 2011, pp. 43-48

[88] Y. Fei, L. Zhong and N. K. Jha, “An energy-aware framework for dynamic software management in mobile computing systems,” in *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 27, pp. 27-56, May 2008

[89] Z. He, W. Cheng and X. Chen, “Energy minimization of portable video communication devices based on power-rate-distortion optimization,” in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, pp. 596-608, May 2008

[90] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano and V. Zaccaria, “Linking run-time resource management of embedded multi-core platforms with automated design-time exploration,” in *Computers & Digital Techniques, IET*, vol. 5, no. 2, pp. 123-135, Mar. 2011

[91] R. Cornea, S. Mohapatra, N. Dutt, A. Nicolau and N. Venkatasubramanian, “Managing cross-layer constraints for interactive mobile multimedia,” in *Proceedings of the 2003 IEEE Workshop Constraint-Aware Embedded Software*, 2003