Optimization of Discrete-parameter Multiprocessor Systems using a Novel Ergodic Interpolation Technique

Neha V. Karanjkar and Madhav P. Desai
Department of Electrical Engineering, Indian Institute of Technology Bombay
e-mail: {nehak,madhav}@ee.iitb.ac.in

Abstract—Design space exploration of multiprocessor systems involves the optimization of cost/performance functions over a large number of design parameters, most of which are discrete-valued. This optimization is non-trivial because the evaluation of cost/performance functions is computationally expensive, typically involving simulation of long benchmark programs on a cycle-accurate model of the system. Further, algorithms for optimization over discrete parameters do not scale well with the number of parameters. We describe a new approach to this optimization problem, based on embedding the discrete parameter space into an extended continuous space. Optimization is then carried out over the extended continuous space using standard descent based continuous optimization schemes. The embedding is performed using a novel simulation-based ergodic interpolation method that produces the interpolated value in a single simulation run. The post-embedding performance function is continuous, and in practice, we observe that it is piecewise smooth. We demonstrate the approach by considering a multiprocessor design exploration problem in which the system being optimized (an eight-core NUMA multiprocessor) has 31 discrete parameters. NAS benchmark kernels are used as workload. The objective function is a weighted sum of cost and performance metrics, and cost-performance trade-off curves are obtained by varying the weights. We use the COBYLA implementation from the Python SciPy library to perform the optimization on the extended continuous space. Near optimal solutions are obtained within three hundred simulation runs, and we observe improvements in the objective function ranging from 1.3X to 12.2X (for randomly chosen initial parameter values). Cost-performance trade-off curves generated from these optimization runs provide clear indicators for the optimal system configuration. Thus, continuous curves from these optimization runs provide clear indicators for the optimal system configuration. Thus, continuous embeddings of discrete parameter optimization problems offer an effective mechanism for the design space exploration of multiprocessor systems.

I. INTRODUCTION

Modern multiprocessor systems have complex architectures, containing multiple components such as cores, caches and interconnects interacting with each other in intricate ways. Designing such a system for optimal performance is non-trivial. Design Space Exploration (DSE) refers to the process of identifying good designs prior to implementation [1]. DSE involves selecting values of a large number of design parameters, most of which are discrete-valued, to optimize cost/performance measures such as execution time and energy consumption under given constraints. The set of all possible values that system parameters can take is referred to as the design space. This is a multi-dimensional space with each dimension corresponding to a design parameter. Cost/performance measures to be optimized over the design space constitute the objective function. The optimization process is non-trivial for two reasons:

1) Cost/performance measures cannot be expressed as a function of design parameters accurately using simple analytical expressions. Simulation of representative benchmark programs on a cycle-accurate model of the system is typically used to evaluate these measures with reasonable accuracy. Evaluating each design option is thus computationally expensive.
2) The design space has a large number of dimensions. Number of possible design options grows exponentially with the number of dimensions.

Techniques for design space exploration aim to find good solutions whilst minimizing the computational expense of finding them. Computational expense for evaluating the objective function at a single point is determined by the level of abstraction of the system model chosen. Hardware prototypes or FPGA implementations provide performance measures with high accuracy but involve very long implementation time, while purely analytical models allow faster evaluation but lose out on accuracy. Simulation based evaluation lies between the two extremes. Our work focusses on exploring the design space efficiently, assuming that the objective function is evaluated using cycle-accurate simulations. Existing techniques for exploring the design space can be broadly classified as follows:

- Exhaustive enumeration: exhaustive search based methods [2] yield globally optimal solutions but the number of evaluations becomes prohibitive for large number of parameters.
- Design of experiments (DoE): number of evaluations can be reduced by carefully selecting a subset of points in the design space to be evaluated, using design of experiments (DoE) approach [3], [4]. However, effectively using DoE approaches other than full-factorial requires prior knowledge about effect of system parameters on performance.
- Search over discrete parameter space: randomized search methods such as simulated annealing [5], [6], evolutionary algorithms [7], [8] and heuristic-based local search methods such as hill climbing [9] and Tabu search [10].
have been applied to cope with the large dimensionality of the design space.

- **Meta-model based search**: Using systematic sampling, a meta model of the system is constructed. The meta model may be used to prune the design space initially, or used in an interleaved manner with simulations to guide the search [11]-[13].

## A. Main Contributions

Most of the parameters in a multiprocessor DSE problem are discrete-valued (for example architectural parameters such as cache dimensions, processor issue-width, buffer sizes and component latencies in units of clock cycles, or software parameters such as mapping of tasks to cores). Existing DSE techniques search for the optimum either directly over the discrete parameter space, or search over a meta-model which may be defined over continuous space.

We describe a new approach for exploring the design space which is based on embedding discrete parameters of the simulation model into continuous space. Descent-based continuous optimization techniques can then be applied directly over the simulation model for finding local optima efficiently. Embedding is performed using a novel simulation-based ergodic interpolation method that produces the interpolation result in a single simulation run. Using continuous optimization techniques offers the following advantages:

1. Continuous optimization methods can handle large dimensionality of the design space better than exhaustive search or design of experiments-based methods. Number of function evaluations is weakly dependent on the number of dimensions.
2. They make use of gradient information and are thus more efficient as compared to randomized search methods for finding local minima.
3. Continuous space offers more pathways to reach the solution as compared to a discrete space. Continuous optimization techniques can recognize diagonal ridges in the objective function unlike local search methods in discrete space such as hill climbing [1].
4. The approach does not involve use of a meta-model, thus each function evaluation is as accurate as the detailed simulation model.

However, descent-based continuous optimization techniques find local minima, and random restarts may have to be used to search for the global optimum. Further, in order to convert the continuous space solution back to discrete space, rounding needs to be employed. This may not always be straightforward. The idea of applying continuous optimization techniques to solve a discrete optimization problem has been described in the past in chemistry [14] and applied mathematics [15]. To our knowledge this approach has not been investigated for system-level design exploration. Also, our embedding technique is highly efficient as compared to standard interpolation methods, which makes this approach practical.

We demonstrate the optimization approach by considering a multiprocessor design exploration problem with 31 discrete parameters. The discrete parameter space is embedded into a continuous space using ergodic interpolation. An implementation of continuous optimization algorithm COBYLA [16] from Python’s SciPy library is used for optimization over the extended continuous space. The objective function is a weighted sum of cost and performance metrics, and cost-performance trade-off curves are obtained by varying the weights. Further, to search for global optimum we perform multiple runs of continuous optimization starting from random initial parameter values. We find that this approach yields near optimal solutions within 300 function evaluations per run. We compare the quality of solutions to those generated by an adaptive simulated annealing (ASA) search in discrete space that is allowed to run for a larger number of iterations. We find that the best solution generated among COBYLA runs is close to the global optimum reported by ASA. We describe the optimization results in Section IV.

## II. System Model

The system being optimized is an eight core multiprocessor system as shown in Figure 1. The model is representative of current multiprocessor NUMA (Non Uniform Memory Access) architectures such as those based on the Intel QPI [17] or AMD HyperTransport [18] standards. The system consists of two processors, each with four cores forming a two-way NUMA configuration. Each core implements the Sparc V8 instruction set. Timing of load/store accesses flowing through the memory subsystem is modeled in detail. All other instructions are assumed to execute in one cycle. The cache subsystem comprises per-core split L1 and unified L2 caches and a shared L3 cache. Coherency is maintained using a hierarchical directory-based MESI protocol which is implemented by generalizing the protocol described in [19], Ch. 8.3.2, p.152) to an arbitrary number of levels in memory hierarchy. Interconnect between successive levels in the memory hierarchy is a full-crossbar with parametrized link delays. The NUMA effect is modeled by assigning different delays to links connecting a processor to its local and remote memory nodes.

A parametrized cycle-accurate model of this system is implemented using the SITAR modeling framework [20].
model is capable of running user-level C programs and parallel applications can be ported to it using a library of synchronization routines. The model has been validated thoroughly using unit tests and large benchmarks.

### A. Embedded Parameters

Although we use functional models for all components in the system, we present an alternate view of the model in order to describe the parameters that are to be embedded into continuous space. The system can be thought of as being composed of three basic components: **modules**, **wires** and **queues** as illustrated in Figure 2. The activity in the system (for example, memory accesses and coherence requests) is modeled by **jobs** and movement of **data-tokens**. A job represents a behavioral action which can consume and produce data-tokens. Data-tokens are used to encapsulate information, and **behavioural action** which can be produced by a single simulation run. The ergodic interpolation as a means of embedding is computationally inefficient for simulation-based optimization. Instead, we introduce an **ergodic interpolation** method which relies on a randomization of the simulation model in order to construct the function \( \hat{\theta}(Y, \Omega_D) \). Spatial interpolation (performed using standard multivariate interpolation methods [21]) is an obvious candidate for \( \theta \). In this paper, we illustrate our approach over the \( C, N, D, L \) parameters introduced above.

### III. Embedding Discrete Parameter Space into Continuous Space

Let \( X = \{x_1, x_2, \ldots, x_n\} \) be a vector of values of discrete-valued design parameters in the model. \( X \in \Omega_D \) where \( \Omega_D \) is the discrete parameter space. Our cycle based simulation model allows us to evaluate some objective function

\[
 f : \Omega_D \rightarrow \mathbb{R}.
\]

The function \( f \) needs to be optimized. We construct an extension of \( f \) to produce a continuous function \( \hat{f} \):

\[
 \hat{f} : \Omega_C \rightarrow \mathbb{R}, \quad \Omega_D \subset \Omega_C \subseteq \mathbb{R}^n
\]

where \( \Omega_C \) is a continuous space extension of \( \Omega_D \). The extension \( \hat{f} \) must satisfy

\[
 \hat{f}(Y) = \begin{cases} 
 f(Y) & \text{when } Y \in \Omega_D \\
 \theta(Y, \Omega_D) & \text{otherwise (1)} 
\end{cases}
\]

That is, \( \hat{f} \) must be continuous in \( \Omega_C \) and must agree with \( f \) on \( \Omega_D \). Thus, we need a suitable **interpolator** \( \theta(Y, \Omega_D) \).

Spatial interpolation (performed using standard multivariate interpolation methods [21]) is an obvious candidate for \( \theta \). In spatial interpolation, for each \( Y \in \Omega_C \), we identify a set of nearest neighbours \( X_1(Y), X_2(Y), \ldots, X_k(Y) \) of \( Y \) such that \( X_i(Y) \in \Omega_D \) for each \( i \). Then,

\[
 \theta(Y, \Omega_D) = I(X_1(Y), X_2(Y), \ldots, X_k(Y))
\]

where \( I \) is some interpolation function [21]. The interpolated value at a single point \( Y \) is then computed in terms of the function values of a set of neighbour points, which have to be computed using expensive simulations. Thus, spatial interpolation as a means of embedding is computationally inefficient for simulation-based optimization.

Instead, we introduce an **ergodic interpolation** method which relies on a randomization of the simulation model in order to construct the function \( \hat{\theta} \). Using this, the value \( \hat{\theta}(Y) \) can be produced by a single simulation run. The ergodic interpolation method builds on the technique described in [22] for producing small (real-valued) perturbations to discrete-valued parameters in a simulation model for measuring sensitivities.
A. Ergodic Interpolation using a Randomized Simulation Model

In order to construct the ergodic interpolator, we first randomize the cycle-based simulation model introduced in Section II. If \( 0 \leq p \leq 1 \) and if \( x \) is a real number, then we define:

\[
\gamma(p, x) = \begin{cases} 
\lfloor x \rfloor & \text{with probability } p \\
\lceil x \rceil & \text{with probability } 1 - p
\end{cases}
\]

Thus, \( \gamma(p, x) \) is an integer-valued random variable. If \( x \) is an integer, then \( \gamma(p, x) = x \). For fixed real \( x \), the expected value of \( \gamma(p, x) \) is \( p \lfloor x \rfloor + (1 - p) \lceil x \rceil \). It follows that for fixed real \( x \), the expected value of \( \gamma(x - \lfloor x \rfloor, x) \) is \( x \).

Suppose the parameters \( C(q), D(j, m), N(m), L(w) \) are real numbers. Then the component behaviour in the simulation model is randomized as follows:

- For a queue \( q \) with real parameter \( C(q) \): Let \( p = C(q) - \lfloor C(q) \rfloor \). At every cycle, accept data-tokens into the queue as long as the total token-weight in the queue is \( \leq \gamma(C(q) - \lfloor C(q) \rfloor, C(q)) \). For example, if \( C(q) = 10.3 \), then \( p = 0.3 \). At each cycle \( \gamma(C(q) - \lfloor C(q) \rfloor, C(q)) \) will be 11 with probability 0.3 and 10 with probability 0.7, so that during the simulation, the queue will have capacity 10 for 70% of the time and capacity 11 for the remaining 30% of the time.

- For a module with real parameter \( N(m) \): At every cycle, start a new job in the module only if the total number of active jobs in the module is less than \( \gamma(N(m) - \lfloor N(m) \rfloor, N(m)) \).

- For a module \( m \) and parameter \( D(j, m) \) for some job \( j \): At every cycle, if the job \( j \) is started successfully, assign a latency of \( \gamma(D(j, m) - \lfloor D(j, m) \rfloor, D(j, m)) \) to the job.

- For a wire \( w \) with real parameter \( L(w) \): At every cycle, for a data-token that enters the wire in this cycle, assign a transport latency of \( \gamma(L(w) - \lfloor L(w) \rfloor, L(w)) \) to the token.

This randomization effectively ensures that the average value of each parameter can be a real number, while the simulation model continues to be discrete parameter and cycle-based. The net effect is that each parameter in the simulation model can be treated as a discrete valued Bernoulli random variable whose time-average value is the desired continuous value at which the function is to be computed. We call this an ergodic interpolation because the time-average in a single simulation run gives the interpolated value.

The rest of the embedding is easy. We embed \( \Omega_D \) into a box \( \Omega_C \) as follows: for each parameter \( p_i \) in the parameter space, we define a minimum possible value \( m_i \) and a maximum possible value \( M_i \). Then:

\[
\Omega_C = \{(x_1, x_2, \ldots, x_n) : m_i \leq x_i \leq M_i, i = 1, 2, \ldots, n\}
\]

For each point \( Y \in \Omega_C \), the ergodic interpolation \( \theta(Y, \Omega_D) \) for \( Y \in \Omega_C \) is produced by the randomized simulation model described above. This technique gives a well defined interpolation. However there are some questions:

1) What is the amount of statistical error in the interpolated value?
2) Is the interpolation well-behaved? That is, is the interpolated function smooth enough for us to be able to use continuous optimization techniques?

We address these questions in the following subsections.

B. Statistical Error in Ergodic Interpolation

Statistical error in the interpolated value can be controlled by increasing the number of samples of parameter values. This can be done by averaging results from multiple simulation runs, or by using a single long simulation run.

For the benchmark programs used as workload in our design exploration experiment (listed in Table I), we estimated the standard deviation of the interpolated value at a few points in the design space by generating multiple samples. For these medium sized benchmarks (spanning 8 to 20 million simulated cycles) we find the standard deviation relative to the mean to be between 0.009% to 0.019%. These error values are small, and thus, for long enough benchmarks, a single simulation run is sufficient for obtaining the interpolated value at a single point.

C. Well-behavedness of Ergodic Interpolation

We check whether the interpolated performance function is smooth and amenable to application of continuous optimization techniques, by evaluating the function at closely-spaced points along 10 random straight lines passing through the parameter space. Each line is sampled at 200 uniformly spaced points. The interpolated function \( \hat{f} \) is the total time to execute the workload. The workload we choose is a memory test program that causes each core to access non-overlapping but interleaved memory locations, and is chosen to stress the memory system sufficiently. Parameters for this experiment are \( D, N \) and \( C \) (output buffers) in L1, L2, L3 caches and main memory (a total of 12 parameters). We measure 2 to 5 samples of \( \hat{f} \) at each point using distinct randomization seeds to compute the mean and relative standard error values. In figure 3, we show the interpolated performance function (the execution time in clock cycles) plots along two of the ten sample lines. We observe that in each case, the interpolated function is continuous and piecewise smooth. Relative standard error values with 2 to 5 samples per point are less than 0.01%. The behaviour of the interpolated performance function along the remaining eight lines is similar. Thus the interpolated functions obtained through the randomized simulation model seem to be well-behaved and suitable for application of continuous optimization techniques.

IV. RESULTS OF CONTINUOUS OPTIMIZATION USING ERGODIC INTERPOLATION

We demonstrate our optimization approach on a multiprocessor design exploration problem with 31 parameters. The system being optimized is described in Section II and the design parameters and their ranges are listed in Table II. Four kernels from the NAS parallel benchmark suite (NPB) [23] are
used as workload. We have ported an OpenMP+C version of NPB v2.3 developed by the Omni project [24] to our model. The kernels and their problem sizes are listed in Table I.

### TABLE I

**NAS Kernels and Their Problem Sizes**

| Kernel                        | Problem Size |
|-------------------------------|--------------|
| Embarrassingly Parallel (EP)  | 2\(^{16}\)   |
| Multigrid (MG)                | 16\(^{4}\)   |
| 3-D FFT PDE solver (FT)       | 16\(^{4}\)   |
| Integer Sort (IS)             | 2\(^{16}\)   |

The discrete parameters listed in Table II are embedded into continuous space using ergodic interpolation. Continuous optimization is performed over this extended continuous space using an implementation of a derivative-free continuous optimization algorithm COBYLA [16] from Python’s SciPy library. To search for the global optimum, we perform multiple runs of COBYLA starting from randomly chosen initial parameter values.

### A. The Objective Function

The objective function is a weighted sum of cost and performance metrics, and cost/performance trade-off curves are obtained by varying the weights. The objective function \( \hat{f} \) is defined as:

\[
\hat{f}(Y) = \text{execution\_time}(Y) + \alpha \times \text{cost}(Y)
\]

where \( Y \) is a vector of parameter values in the extended continuous space, \( \alpha \) is a constant representing weight assigned to cost, \( \text{execution\_time}(Y) \) is the sum of execution times for four benchmark kernels (listed in Table I) running on the simulation model, and \( \text{cost}(Y) \) is a synthetic cost function which increases as each parameter is varied in the direction of improving performance. The cost function is defined as:

\[
\text{cost}(Y) = \sum_{i} x_i + \sum_{j} \frac{d_j}{\alpha_i}
\]

where \( d_j \) and \( x_i \) are delay and parameter values to get cost/performance trade-off curves. Further, for each value of \( \alpha \), multiple optimization runs starting from eight distinct randomly chosen points in the parameter space are performed to search for the global optimum. A single optimization run is allowed to make at most 300 function evaluations. In Figure II we show the cost and performance values at the optimum reported by all simulation runs. The plot shows a clear knee which can be used to select the optimal system configuration for maximum performance. Parameter values at the best solution among all COBYLA runs (for \( \alpha = 10^5 \)) at the knee are listed in Table III.

### B. Results

Optimization is performed with multiple values of the weight factor \( \alpha \in \{0, 10^4, 10^5, 10^6\} \) to get cost/performance trade-off curves. Further, for each value of \( \alpha \), multiple optimization runs starting from eight distinct randomly chosen points in the parameter space are performed to search for the global optimum. A single optimization run is allowed to make at most 300 function evaluations. In Figure II we show the cost and performance values at the optimum reported by all simulation runs. The plot shows a clear knee which can be used to select the optimal system configuration for maximum performance. Parameter values at the best solution among all COBYLA runs (for \( \alpha = 10^5 \)) at the knee are listed in Table III.

In Table III we list the best, average and worst case improvements in objective function values over the initial guess among eight optimization runs for each value of \( \alpha \).

### TABLE III

**Improvements in Objective Function Value Over the Initial Guess**

| \( \alpha \)         | Best-case | Average | Worst-case |
|----------------------|-----------|---------|-----------|
| \( \alpha = 0 \)     | 2.8x      | 1.8x    | 1.4x      |
| \( \alpha = 10^4 \)  | 2.5x      | 2.8x    | 1.3x      |
| \( \alpha = 10^5 \)  | 3.3x      | 2.8x    | 2.1x      |
| \( \alpha = 10^6 \)  | 12.2x     | 8.8x    | 4.4x      |

We compare the quality of solutions generated by COBYLA to those generated by an Adaptive Simulated Annealing (ASA) search over discrete parameter space. We use a Python binding...
of a well-established ASA implementation. Table IV shows the best-case, worst-case and average-case values of objective function at the optimum as reported by COBYLA (with 300 function evaluations) and the optimum reported by ASA (with 1000 function evaluations). It is evident that the COBYLA optimization of the interpolated objective function produces near optimal solutions (if we assume that a long running ASA leads us to the optimal solution).

TABLE IV

OBJECTIVE FUNCTION VALUES AT THE OPTIMUM AS REPORTED BY COBYLA AND ASA

|          | objective function values at the optimum |          |          |          |
|----------|-----------------------------------------|----------|----------|----------|
|          | \(\alpha = 0\)                          | \(\alpha = 10^4\) | \(\alpha = 10^5\) | \(\alpha = 10^6\) |
| COBYLAworst | 2.926 \(\times 10^7\)                  | 4.020 \(\times 10^7\) | 7.401 \(\times 10^7\) | 2.836 \(\times 10^8\) |
| COBYLAaverage | 2.918 \(\times 10^7\)                  | 3.837 \(\times 10^7\) | 6.305 \(\times 10^7\) | 1.608 \(\times 10^8\) |
| COBYLANbest | 2.916 \(\times 10^7\)                  | 3.700 \(\times 10^7\) | 5.753 \(\times 10^7\) | 1.218 \(\times 10^8\) |
| ASA       | 2.916 \(\times 10^7\)                  | 3.694 \(\times 10^7\) | 6.063 \(\times 10^7\) | 1.247 \(\times 10^8\) |

V. Conclusions

We have described a technique using which discrete parameter multiprocessor systems can be optimized using continuous space optimization schemes. The technique relies on a novel ergodic interpolation scheme based on a randomized discrete parameter and cycle based simulation model of the multiprocessor system. For a representative set of large benchmark programs, the resulting interpolated function has low statistical error, is continuous and piecewise smooth. We have applied a standard optimization algorithm to find optimal solutions for a 31-parameter multiprocessor system exercised with a subset of the NAS benchmarks. The optimization algorithm applied to these benchmarks produces substantial improvements ranging from 1.3X to 12.2X over the initial guess in the cases that we have tried. Cost performance curves can also be generated using different weightings of the performance and cost functions in the objective function. We conclude that ergodic interpolation based optimization is an effective and practical approach for design space exploration of multiprocessor systems.

ACKNOWLEDGMENT

Part of this work was funded by IBM grant 11IBM002. Simulations in Section III-C were run on CDAC’s PARAM Yuva II cluster. The authors wish to thank prof. Virendra Sule for granting us the use of cluster facilities and prof. Sachin Sapatnekar for his suggestions and feedback.

REFERENCES

[1] M. Gries, “Methods for Evaluating and Covering the Design Space during Early Design Development,” Integration, the VLSI journal, vol. 38, no. 2, 2004.
[2] T. Givargis, J. Henkel, and F. Vahid, “Interface and Cache Power Exploration for Core-based Embedded System Design,” in 1999 IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers. IEEE, 1999.
[3] D. Sheldon, F. Vahid, and S. Lonardi, “Soft-core Processor Customization using the Design of Experiments Paradigm,” in 2007 Design, Automation & Test in Europe Conference & Exhibition. IEEE, 2007.
[4] J. Yi, D. Lilja, and D. Hawkins, “A Statistically Rigorous Approach for Improving Simulation Methodology,” in The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings. IEEE Comput. Soc, 2003.
[5] H. Orsila, E. Salminen, M. Hannikainen, and T. D. Hamalainen, “Evaluation of Heterogeneous Multiprocessor Architectures by Energy and Performance Optimization,” in 2008 International Symposium on System-on-Chip. IEEE, 2008.
[6] B. C. Schafer, “Adaptive Simulated Annealer for High Level Synthesis Design Space Exploration,” in 2009 International Symposium on VLSI Design, Automation and Test. IEEE, 2009.
[7] M. Holzer, B. Knerr, and M. Rupp, “Design Space Exploration with Evolutionary Multi-Objective Optimization,” in 2007 International Symposium on Industrial Embedded Systems. IEEE, 2007.
[8] M. Palesi and T. Givargis, “Multi-objective Design Space Exploration using Genetic Algorithms,” in Proceedings of the Tenth International Symposium on Hardware/Software Codesign. CODES 2002. ACM, 2002.
[9] K. Lahiri, A. Ragunathan, and S. Dey, “Efficient Exploration of the SoC Communication Architecture Design Space,” in IEEE/ACM International Conference on Computer Aided Design. ICCAD - 2000. IEEE/ACM Digest of Technical Papers. IEEE, 2000.
[10] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, “System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search,” Design Automation for Embedded Systems, vol. 2, no. 1, 1997.
[11] G. Palermo, C. Silvano, and V. Zaccaria, “An Efficient Design Space Exploration Methodology for Multiprocessor SoC Architectures based on Response Surface Methods,” in 2008 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation. IEEE, Jul. 2008.
[12] R. Piscitelli and A. D. Pimentel, “Design Space Pruning through Hybrid Analysis in System-level Design Space Exploration,” in 2012 Design, Automation & Test in Europe Conference & Exhibition. IEEE, Mar. 2012.
[13] E. Ipek, S. A. McKee, R. Caruana, B. R. de Supinski, and M. Schulz, “Efficiently Exploring Architectural Design Spaces via Predictive Modeling,” ACM SIGARCH Computer Architecture News, vol. 34, no. 5, Oct. 2006.
[14] S. K. Koh, G. Ananthasuresh, and S. Vishveshwara, “A Deterministic Optimization Approach to Protein Sequence Design Using Continuous Models,” The International Journal of Robotics Research, vol. 24, no. 2-3, Feb. 2005.
[15] H. Wang and B. W. Schmeiser, “Discrete Stochastic Optimization using Linear Interpolation,” in 2008 Winter Simulation Conference. IEEE, Dec. 2008.
[16] M. Powell, “On Trust Region Methods for Unconstrained Minimization without Derivatives,” Mathematical Programming, vol. 97, no. 3, 2003.
[17] D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek, “Intel Quick-Path Interconnect Architectural Features Supporting Scalable System Architectures,” in 2010 18th IEEE Symposium on High Performance Interconnects. IEEE, 2010.

[18] C. Keltcher, K. McGrath, A. Ahmed, and P. Conway, “The AMD Opteron Processor for Multiprocessor Servers,” IEEE Micro, vol. 23, no. 2, Mar. 2003.

[19] D. Sorin, M. Hill, and D. Wood, A Primer on Memory Consistency and Cache Coherence. Morgan and Claypool Publishers, 2011.

[20] N. V. Karanjkar and M. P. Desai, “SiTAR : Simulation Tool for Architectural Research. Technical report,” 2012, unpublished.

[21] B. Bojanov, H. Hakopian, and B. Sahakian, Spline Functions and Multivariate Interpolations, ser. Mathematics and Its Applications. Springer, 1993.

[22] G. Hazari, M. P. Desai, and G. Srinivas, “Bottleneck Identification Techniques Leading to Simplified Performance Models for Efficient Design Space Exploration in VLSI Memory Systems,” in 2010 23rd International Conference on VLSI Design. IEEE, 2010.

[23] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, “The NAS Parallel Benchmarks Summary and Preliminary Results,” in Proceedings of the 1991 ACM/IEEE Conference on Supercomputing, ser. Supercomputing ’91. New York, NY, USA: ACM, 1991.

[24] K. Kusano, S. Satoh, and M. Sato, “Performance Evaluation of the Omni OpenMP Compiler,” in High Performance Computing, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, vol. 1940.

[25] J. Robert. Python bindings for the asa code. [Online]. Available: https://pypi.python.org/pypi/pyasa/

[26] L. Ingber, “Adaptive Simulated Annealing (ASA): Lessons Learned,” Control and Cybernetics, vol. 25, 1996.