An Approach to Discrete Parameter Design Space Exploration of Multi-core Systems using a Novel Simulation Based Interpolation Technique

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

***Abstract*—We propose a new approach to simulation-based design optimization of multi core systems, over a large number of discrete parameters. In this approach, we embed the discrete parameter space into an extended continuous space and apply continuous space optimization techniques over the embedding to search for optimal designs. Such continuous space techniques often scale well with the number of parameters. The embedding is performed using a novel simulation-based ergodic interpolation technique, which, unlike spatial interpolation methods, can produce the interpolated value within a single simulation run irrespective of the number of parameters. In a characterization study, we find that the interpolated performance curves are continuous, piecewise smooth and have low statistical error. We use the ergodic interpolation-based approach to solve a multi-core design optimization problem with 31 design parameters. Our results indicate that continuous space optimization using ergodic interpolation-based embedding can be a viable approach for large multi-core design optimization problems.**

I. Introduction

Design space exploration of multi-core systems involves optimizing performance/cost over a large number of design parameters, most of which are discrete-valued (for example, buffer sizes, associativity and size of caches, number of pipelined stages in interconnects etc.). This optimization is hard because the accurate evaluation of a potential design choice is computationally expensive, involving detailed cycle-accurate system simulation to predict performance metrics such as execution time and energy consumption [1]. Further, the number of design parameters is likely to keep increasing as chip complexity rises. Existing techniques for multi-core design exploration involve searching for optimal designs directly over the discrete parameter space (for example exhaustive enumeration, design of experiments, randomized search methods such as simulated annealing and genetic algorithms), *or* use a meta-model of the system (in the form of an artificial neural network, regression model or polynomial interpolation etc.), constructed by systematically sampling the design space, to guide the search during optimization.

If the discrete parameter space can be *embedded* into a larger continuous parameter space, then continuous space techniques can, in principle, be applied to the system optimization problem. Such continuous space techniques often scale well with the number of parameters. However, descent-based continuous optimization methods find local minima and random restarts may be needed to search for the global optimum. Further, in order to convert the continuous-space solution back to discrete-space, rounding needs to be employed with care. The idea of applying continuous space optimization to solve discrete optimization problems using an embedding, has been described in the past in chemistry [2] and applied mathematics [3]. To our knowledge this approach has not been investigated for computer system optimization.

We present a novel simulation-based *Ergodic Interpolation* technique for embedding the discrete parameter space into a continuous space efficiently. The technique is based on randomization of the simulation model in order to obtain an interpolation of performance/cost functions, and, unlike spatial interpolation methods, can produce the interpolated value at a point within a single simulation run. We describe the technique in Section III. We have implemented this interpolation scheme in a cycle-accurate multi-core system simulator, and characterized it on a problem instance with twelve design variables. We observe that the statistical error in the interpolation is small, and the interpolated function is continuous and piecewise smooth.

We validate the ergodic interpolation-based optimization approach on a large multi-core design optimization problem with 31 discrete parameters. The system being optimized is an eight-core NUMA multiprocessor running the NAS benchmark kernels. The objective function to be minimized is a weighted sum of cost and performance metrics where weights are varied to get cost-performance trade-off curves. Continuous optimization (using an implementation of COBYLA from Python's SciPy library) is performed over the embedded objective function. Starting from multiple random initial points, we ran the optimization algorithm with a limit of 300 objective function evaluations per optimization run. The solution converged to a local optimum in all cases and the improvement in the objective ranged from 1.3X to 12.2X over the initial guess across multiple runs. Further, the spread in objective values at the optimum, across multiple runs was low. Cost-performance trade-off curves generated from these optimization runs provide clear indicators for the optimal system configuration. The results (presented in Section IV) indicate that continuous space optimization using ergodic interpolation-based embedding can be a viable approach to solve design exploration problems with several discrete parameters.

1526-7539/15 $31.00 © 2015 IEEE
DOI 10.1109/MASCOTS.2015.26

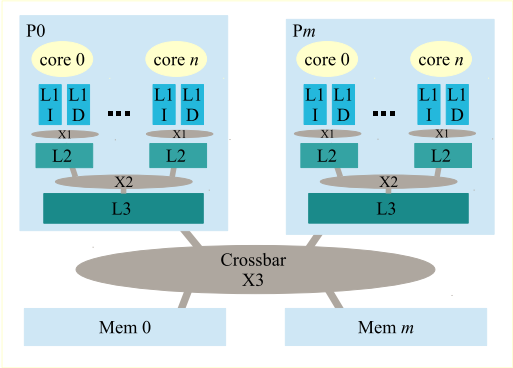

Fig. 1. System Model

II. System Model and Workload

For all simulations presented in this study, we use a cycle-accurate simulation model of a multi-core, multi-processor NUMA (Non Uniform Memory Access) system as shown in Figure 1. The system consists of m processors, with n cores per processor. Each core implements the Sparc V8 instruction set. Timing of load/store accesses flowing through the memory subsystem is modeled in detail and other instructions are assumed to execute in one cycle. The cache subsystem comprises per-core split L1 I/D and unified L2 caches and a shared L3 cache. Coherency is maintained using a hierarchical directory-based MESI protocol. 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. For all simulations we have used $m = 2$ and $n = 4$. For the optimization study presented in Section IV we have used four kernels from the NAS parallel benchmark suite (NPB) [4] (listed in Table I) as workload.

TABLE I
NAS benchmark kernels and problem sizes

Embarrassingly Parallel	2^{16}	3-D FFT PDE solver	16^3
Multigrid	16^3	Integer Sort	2^{16}

A. Embedded Parameters

We consider four types of discrete-valued parameters for a preliminary validation of our optimization approach. We give a precise definition of these parameters in this section. The system can be thought of as being composed of three basic components: *modules*, *wires* and *queues*. Modules represent behavioral components in the system such as caches, cores, memory modules and interconnect schedulers, wires represent interconnect links with a parametrizable number of pipelined stages, and queues are used to represent buffering at various places in the system. 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 behavioural action by a module which can consume and produce data-tokens. Data-tokens are used to encapsulate information. Jobs are triggered inside modules by the availability of the necessary data-tokens. The completion of a job may produce new data-tokens. Queues are used to buffer data-tokens. Each module m, wire w and queue q in the system has the following parameters :

- **queue capacity** $C(q)$ **:** Each queue/buffer in the system q has a single parameter $C(q)$. In each cycle, the queue q will accept new data-tokens as long as the total number of data-tokens in the queue is $\leq C(q)$.
- **module throughput** $N(m)$ **:** a module m can accept new jobs each cycle as long as the number of jobs being processed by it is $\leq N(m)$.
- **module delay** $D(j, m)$ **:** If j is a job that is accepted by module m, then $D(j, m)$ is the number of cycles it takes for the module to execute the job. The job is removed from the input queue as soon as it is accepted by the module and place is reserved for its output in the output queues. Its output becomes visible in the output queues only after $D(j, m)$ cycles.
- **wire latency** $L(w)$ **:** A wire w has a single parameter $L(w)$ that represents the number of register stages in the wire. A wire can accept at most one data-token each cycle, and each token takes $L(w)$ cycles to pass through the wire. The wire can accept a token only after reserving a place for it in the output queue to which it is connected. The token becomes visible in the output queue after $L(w)$ cycles.

To summarize: our cycle accurate simulation model has the discrete parameters $C(q)$, $N(m)$, $D(j, m)$ and $L(w)$ for each queue q, module m and wire w in the system. We describe an embedding for these parameters in the next section.

III. Ergodic Interpolation

Consider a multi-core system with n discrete-valued design parameters. Let $X = (x_1, x_2, ...x_n)$ be a vector of parameter values. $X \in \Omega_D$ where Ω_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.

Our approach is to extend f to produce a continuous function $\hat{f}$ which is an *interpolation* of f defined over a continuous space, and then apply continuous space optimization techniques to minimize $\hat{f}$. Thus we wish to find an interpolation $\hat{f} : \Omega_C \rightarrow \mathbb{R}$ where Ω_C is the extended continuous parameter space ($\Omega_D \subset \Omega_C \subseteq \mathbb{R}^n$). Spatial interpolation (performed using standard multivariate interpolation methods such as Lagrange interpolation, Simplex interpolation etc.) is inefficient for the purpose of obtaining $\hat{f}$, because evaluating $\hat{f}(Y)$ at each point requires multiple expensive simulations.

Instead, we propose an *ergodic interpolation* method which can produce the value $\hat{f}(Y)$ at a point $Y \in \Omega_C$ within a single simulation. The method relies on a randomization of the simulation model to obtain $\hat{f}(Y)$ directly. It builds on a sensitivity measurement technique described in [5] for producing small (real-valued) perturbations to discrete-valued parameters in a simulation model. Let $Y = (y_1, y_2, \ldots, y_i, \ldots, y_n)$ be a

point in the continuous parameter space ($Y \in \Omega_C$), at which we wish to evaluate the interpolated objective function $\hat{f}(Y)$. We replace each discrete parameter i in the cycle-accurate simulation model with a discrete random variable, whose value changes over time within a single simulation run, such that its *average* value over the entire simulation approaches $y_i \in Y$. The set of time-averaged values of all parameters approach $(y_1, y_2, \ldots, y_n)$ and the cost/performance values obtained by simulating this randomized model gives us the interpolated value $\hat{f}(Y)$. Thus, unlike spatial interpolation methods which perform an averaging in space, ergodic interpolation works by *averaging in time*. We define the embedding for $C(q)$, $N(m)$, $D(j,m)$ and $L(w)$ parameters (defined in Section II-A) as follows: Let p be a discrete-valued parameter. Suppose we wish to assign a real value $v \in \mathbb{R}$ to the parameter p in the embedded model. We replace p with an integer-valued Bernoulli random variable $\gamma(v)$, whose value changes *every cycle* with the following distribution:

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

If x is an integer, then $\gamma(x) = x$. If x is a fixed real number, the value of $\gamma(x)$ alternates between $\lfloor x \rfloor$ and $\lceil x \rceil$, such that its expected value is x. We replace each $C(q)$, $N(m)$, $D(j,m)$ and $L(w)$ parameter in the model with random variables $\gamma(C(q))$, $\gamma(N(m))$, $\gamma(D(j,m))$ and $\gamma(L(w))$ respectively. Thus real values can be assigned to the parameters. For example, if $C(q) = 10.3$, then at each cycle $\gamma(C(q))$ will be 11 with probability 0.3 and 10 with probability 0.7, so that, during simulation, the queue will have a capacity 11 for 30% of the time, and a capacity 10 for 70% of the time.

Randomization introduces a statistical error in the measured value of $\hat{f}(Y)$. This error can be controlled by using longer workloads, or by averaging results from multiple simulation runs. For the workloads used in this study, we estimated the standard deviation in $\hat{f}$ at a few points in the design space by generating multiple samples. The standard deviation of $\hat{f}(Y)$ relative to the mean was between 0.009% to 0.019%. These error values are small and therefore a single simulation run was sufficient to measure the interpolated value.

A. Well-behavedness of Ergodic Interpolation

We perform a simulation-based characterization study to check whether the interpolated performance function $\hat{f}(Y)$ is smooth, and suited to continuous space optimization. The objective function $\hat{f}(Y)$ is the total execution time for a parallel memory-test workload evaluated using simulation of the cycle accurate model described in Section II. Design parameters are D, N and C(for output buffers) (as introduced in Section II-A) in L1, L2, L3 caches and main memory. Thus the extended continuous parameter space has 12 dimensions. We consider ten random straight lines passing through the 12-dimensional continuous parameter space. $\hat{f}(Y)$ is sampled at 200 uniformly spaced points along each line. We perform multiple simulation runs at each point with distinct randomization seeds to measure the mean and standard error values in $\hat{f}$. In Figure 2, we plot mean $\hat{f}$ along two of the ten random lines passing through the parameter space. We observe that along each line, the interpolated function is continuous and piece-wise smooth. The plots for other lines are similar. The measured relative standard error values are less than 0.01%. Thus the interpolated function obtained using our ergodic interpolation technique seems to be well-behaved and suitable for the application of continuous optimization techniques.

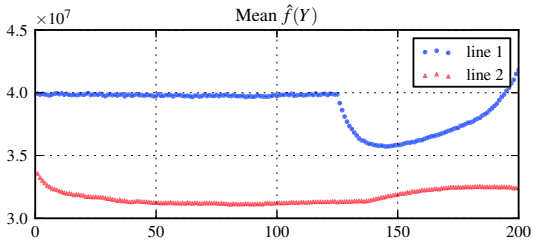

Fig. 2. Mean $\hat{f}(Y)$ values along two random straight lines passing through the multi-dimensional continuous parameter space. Each line is sampled at 200 uniformly spaced points.

IV. Continuous Optimization using Ergodic Interpolation

We validate the ergodic-interpolation based optimization approach on a multi-core design optimization problem with 31 parameters. The design parameters are :

- throughput (N) and delay (D) of L1I, L1D, L2, L3 caches and main memory
- interconnect latency (L) in switches X1, X2 and X3
- input and output queue-sizes (C_{inq} and C_{outq}) in L1I, L1D, L2, L3 caches, main memory and switches X1, X2 and X3[1].

The parameters are embedded into a continuous space using ergodic interpolation. The objective function $\hat{f}(Y)$ is a weighted sum of performance and cost measures. The performance measure execution_time(Y) is the sum of execution times for four benchmark kernels (listed in Table I) evaluated using the cycle-accurate simulation model. We represent cost using a synthetic function cost(Y) which increases as each parameter is varied in the direction of improving performance. The cost function is defined as $\text{cost}(Y) = \sum_i x_i + \sum_j \frac{100}{d_j}$ where d_j and x_i are delay and non-delay parameters normalized to lie in the range $[1, 100]$. The objective function is:

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

Where α is a weighting factor. 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. We use an implementation of COBYLA [6], a derivative-free, noise insensitive continuous space optimization algorithm from Python's SciPy

[1]In the switch X3, paths to local and remote nodes are assigned different latencies (L) and output queue-sizes (C_{outq}) to model the NUMA-effect

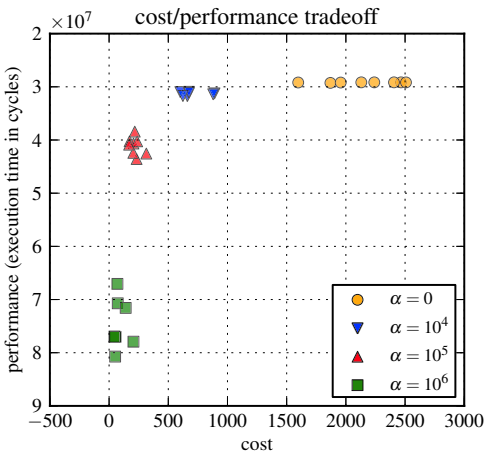

Fig. 3. Performance and cost values at the optimum for multiple optimization runs (as α and initial points are varied). Each point represents the result of a single optimization run with (x,y) coordinates showing (cost, performance) values at the optimum.

library to minimize $\hat{f}(Y)$. Since the algorithm searches for local minima, we use multiple runs of COBYLA starting from distinct randomly-chosen initial points in the design space. For each value of α, we perform eight optimization runs starting from distinct initial points. A single optimization run is allowed to make at most 300 function evaluations.

We observe that for all optimization runs, the objective function values converge to a local optimum within 300 function evaluations. Further, the spread in objective values across optimization runs with different initial points is small for most cases, as summarized in Table II. Improvements over the initial guess range from 1.3X to 12.2X across all optimization runs.

TABLE II
OBJECTIVE FUNCTION VALUES AT THE OPTIMUM FOR EACH VALUE OF α

	$\alpha = 0$	$\alpha = 10^4$	$\alpha = 10^5$	$\alpha = 10^6$
Objective values at the optimum across eight COBYLA runs				
best	2.916×10^7	3.697×10^7	5.753×10^7	1.218×10^8
worst	2.922×10^7	4.020×10^7	7.401×10^7	2.836×10^8
mean	2.917×10^7	3.836×10^7	6.305×10^7	1.608×10^8
std dev	0.08%	3.08%	8.59%	35.66%
Objective values at the optimum found by ASA				
ASA	2.916×10^7	3.694×10^7	6.063×10^7	1.247×10^8

In Figure 3, we plot cost and performance values at the optimum for each of the optimization runs (as α and initial points are varied). Each point in the plot represents the result of a single optimization run with (x,y) coordinates showing (cost, performance) values at the optimum. The plot shows a clear *knee* which can be used to select the optimal system configuration for maximum performance.

We compare the locally-optimum solutions found by COBYLA to a global-optimum produced by an Adaptive Simulated Annealing (ASA) search over discrete parameter space, with limit of 1000 function evaluations. We use a Python binding of a well-established ASA implementation [7]. The objective values at the optimum found by ASA are listed in the last row in Table II. We observe that most of the COBYLA runs produce solutions that are close to (within 10% of) the global optimum reported by ASA.

V. CONCLUSIONS

We have described a technique using which discrete parameter multi-core systems can be optimized using continuous space optimization schemes. The technique relies on a novel ergodic interpolation scheme based on randomizing the discrete parameter cycle-accurate simulation model of the multi-core system. The interpolated performance function is continuous, has low statistical error, and was observed to be piece-wise smooth. Using this ergodic interpolation technique, we have applied a standard continuous space optimization algorithm to find optimal designs for a 31-parameter multiprocessor system exercised with a subset of the NAS benchmarks. The optimization algorithm converged to a local optimum within 300 function evaluations and produced 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 components in the objective function.

More work is needed to completely characterize the impact of rounding on the quality of the results obtained, and on the application of the ergodic interpolation technique to other discrete parameters such as cache size/associativity and processor core issue-width/clock-frequency. However, our preliminary investigations indicate that ergodic interpolation based optimization can be an effective and practical approach for the design space exploration of multi-core systems.

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] 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.
[3] H. Wang and B. W. Schmeiser, "Discrete Stochastic Optimization using Linear Interpolation," in *2008 Winter Simulation Conference*. IEEE, Dec. 2008.
[4] 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.
[5] 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.
[6] M. Powell, "On Trust Region Methods for Unconstrained Minimization without Derivatives," *Mathematical Programming*, vol. 97, no. 3, 2003.
[7] L. Ingber, "Adaptive Simulated Annealing (ASA): Lessons Learned," *Control and Cybernetics*, vol. 25, 1996.