A High-Level Analytical Model for Application Specific CMP Design Exploration

Andrew Cassidy*, Kai Yu†, Haolang Zhou*, and Andreas G. Andreou*
*Department of Electrical and Computer Engineering, Johns Hopkins University, Baltimore, MD 21218
Email: {acassidy, hao, andreou}@jhu.edu
†Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213
Email: kaiy@ece.cmu.edu

***Abstract*—We present a high-level analytical model for chip-multiprocessors (CMPs) that encompasses processors, memory, and communication in an area-constrained, global optimization process. Applying this analytical model to the design of a symmetric CMP for speech recognition, we demonstrate a methodology for estimating model parameters prior to design exploration. Then we present an automated approach for finding the optimal high-level CMP architecture. The result is the ability to find the allocation of silicon resources for each architectural element that maximizes overall system performance. This balances the performance gains from parallelism, processor microarchitecture, and cache memory with the energy-delay costs of computation and communication.**

I. Introduction

As integrated circuit technology steadily progresses, the capability to build chip-multiprocessors (CMPs) with hundreds or thousands of processor cores is imminent. This prospect poses daunting hurdles for the electronic design automation (EDA) industry. With systems of such massive scale and complexity, manual design is unreasonable. Moreover, current design methodologies, such as instruction set simulators (ISS) and cycle accurate simulators, are too detailed to quickly explore the system-level design space. Analytical models are one approach to quickly identifying advantageous architectures.

In this work, we present a high-level analytical model for CMPs that encompasses processors, memory, and communication in an area-constrained, global optimization process. We detail a methodology for estimating the model parameters in order to tailor CMPs to specific applications, as well as an automated method for solving to find the optimal architecture.

Briefly, large vocabulary continuous speech recognition requires significant computational power, even by today's microprocessor standards, to reach real-time recognition rates. Applications benefiting from improved real-time speech recognition performance range from hand-held wireless devices to robots using speech as a human-computer interface to call mining and call monitoring. We use this application domain as an example to demonstrate design exploration and optimization of an application specific CMP.

A. Background

Recently, there have been a variety of high-level analytical models for CMPs. Hill and Marty introduced an analytical model for processor performance and the number of cores in symmetric, asymmetric, and dynamic multicore chips [1]. Another approach [2] extended Hill and Marty's model to include energy. Other research investigates the tradeoff between the number of CMP cores and cache architecture, including parameters such as cache size, L2 and L3 cache effects, shared vs. private vs. hybrid caches, and uniform vs. non-uniform caches [3]. Two other works explore the performance degradation due to inter-thread cache contention in shared cache CMPs [4], [5]. In contrast to all of these works, we present an analytical model that jointly evaluates the tradeoffs between the number of cores, the processor performance, cache memory size (and hit rate), and communications interconnect in an area-constrained, global optimization process.

Others have considered constrained optimization, including power and thermal constraints [6], [7]. These approaches emphasize the importance of joint optimization across interrelated variables and inclusion of constraints during optimization. However, both approaches are based on simulators, instead of analytical models. Our approach incorporates the area constraint directly into the analytical model.

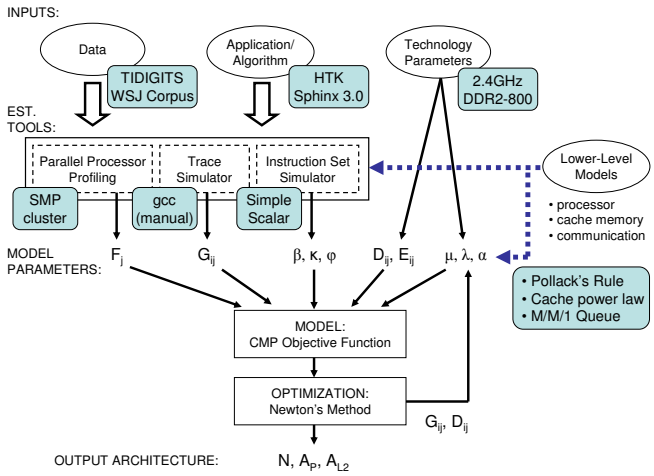

Fig. 1. Methodology overview for optimizing Application Specific CMPs. Actual data, tools, models, and parameters used in our design example are given in the shaded blocks.

II. Methodology

Fig. 1 depicts the overall view of our methodology for optimizing application specific CMPs. Beginning with the

978-3-9810801-7-9/DATE11/©2011 EDAA

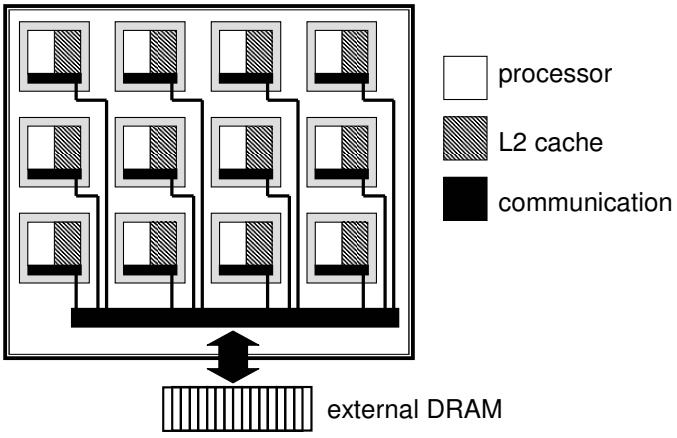

Fig. 2. Symmetric CMP Block Diagram with $N = 12$ cores and a shared memory controller to external DRAM.

four entities given in the ovals, the targeted application or algorithms and data set govern the application characteristics, while the technology parameters and lower-level models govern the architecture characteristics. The lower-level models are parameterized to facilitate customization and optimization of the architecture given the specific application. We combine the input data, algorithms, and technology with the estimation tools, experiments, simulations, and manual analysis in order to estimate the parameter values for the lower-level models. The models and parameters are combined in the CMP objective function. Architectural optimization is performed by minimizing the objective function subject to the area constraint. We use the method of Lagrange multipliers and Newton's method as an automated method to find the minimum cost solution. The optimization is iterated in order to refine the estimate of the queueing model parameter λ. When optimization is finished, we have found the high-level specification of the CMP architecture, namely the number of cores N, the processor core size A_P, and the L2 cache size A_{L2}. The actual data, tools, models, and parameters used in the experiments here are given in the shaded blocks. Each step of this methodology flow is described in detail in the following sections.

III. High Level Model

For optimization, we begin by defining the high-level characteristics of the CMP. In this example, we assume a symmetric CMP, dedicated L1 and L2 caches per core, and a shared interface to main memory. Fig. 2 depicts a block diagram of this configuration for a CMP with twelve cores. Each core has a dedicated communication channel to the memory controller. The memory controller acts as an arbitrator between contending cores for the shared resource (main memory). We proceed to find the number of cores, processor core size/performance, and the L2 cache memory size that maximizes the performance of the CMP for speech recognition.

A. Theory

We use the symmetric CMP objective function for high-level optimization presented in [8]. Summarizing, the objective function $J_{D \cdot E}$ is composed of two parts. The outer summation in the objective function accounts for the parallel performance of the CMP, while the inner summation accounts for the performance of individual components within the architecture.

$$J_{D \cdot E} = \sum_{j=0}^{K-1} F_j N_j^{\gamma-1} \sum_{i=0}^{M-1} G_{ij} D_{ij} E_{ij}^{\gamma} \quad (1)$$

In the outer summation, F_j is the fraction of the algorithm that has parallelism of N_j. In the inner summation, each of the algorithm fractions, F_j, are subdivided into constituent cost components. G_{ij} is the fraction of F_j that has the ith cost component $D_{ij}E_{ij}$. The ith delay is D_{ij} and the ith energy cost is E_{ij} for the jth fraction of the algorithm. F_j and G_{ij} are fractions, such that $\sum_{j=0}^{K-1} F_j = 1$ and $\sum_{i=0}^{M-1} G_{ij} = 1$; The exponent γ weights the effect of energy with respect to delay in the cost function. A value of $\gamma = 1.0$ reflects equal weighting of energy and delay, while a value of $\gamma = 0$ corresponds to no contribution of energy to the cost function[1]. The fractions F_j represent task or thread level parallelism, while parallelism due to data level and instruction level parallelism (ILP) are modeled by the terms in the inner summation ($G_{ij}D_{ij}$).

IV. Lower Level Models

In this section, we specify the lower-level models of processors, memory, and communication to be used in the objective function.

A. Processor Models

Pollack's Rule [9] observes that the performance of a processor is proportional to the square root of the area of the processor, or inversely:

$$CPI \propto A_P^{-\frac{1}{2}} \quad (2)$$

where CPI is the average cycles-per-instruction, a measure of time per instruction. This observation is based on the diminishing effect of microarchitectural techniques, such as those associated with super-scalar microarchitecture (number of arithmetic or logic functional units, instruction issue width, in vs. out of order execution, etc.) Adding a proportionality constant β and an offset constant ϕ_P yields a parameterized equality that we can use as a lower level relationship between processor performance and processor area:

$$CPI = \beta A_P^{-\frac{1}{2}} + \phi_P \quad (3)$$

If CPI is estimated using an ideal memory hierarchy (all instructions and data hit in L1 cache), then CPI is a good

[1]Strictly speaking, $D_{ij}E_{ij}$ is no longer the energy-delay product, rather it is the energy-delay dot product, combining the energy-delay products of the constituent components.

estimate for the time or delay for instructions at the first level of the architecture:

$$D_0 = CPI = \beta A_P^{-\frac{1}{2}} + \phi_P \quad (4)$$

The parameters β and ϕ_P are estimated in Section V.

B. Cache Memory Models

An approximate rule of thumb for caches is that miss rate MR is inversely proportional to the square root of the size (or area, A_{CM}) of the cache memory [10]:

$$MR \quad \propto \quad \frac{1}{\sqrt{A_{CM}}} \quad (5)$$

The basis for this relation was extensively studied by [11]. Adding a proportionality constant κ and an offset constant ϕ_{CM} gives us a parameterized equality that we can use as a lower level relationship between cache memory miss rate and cache memory size:

$$MR \quad = \quad \kappa A_{CM}^{-\frac{1}{2}} + \phi_{CM} \quad (6)$$

The L2 cache memory hit rate HR is:

$$HR_{L2} \quad = \quad 1 - MR_{L2} = 1 - \kappa A_{L2}^{-\frac{1}{2}} - \phi_{L2} \quad (7)$$

The hit rate modulates the fraction of instructions that hit in the register file or L1 cache (G_0), the L2 cache (G_1), and main memory (G_2):

$$\begin{aligned} G_0 &= HR_{RF,L1} && (8)\\ G_1 &= (1 - G_0) HR_{L2} \\ &= (1 - G_0)(1 - \kappa A_{L2}^{-\frac{1}{2}} - \phi_{L2}) && (9)\\ G_2 &= (1 - G_0) MR_{L2}\ HR_{mem} \\ &= (1 - G_0)(\kappa A_{L2}^{-\frac{1}{2}} + \phi_{L2}) 100\% && (10) \end{aligned}$$

Section V details the estimation of the parameters κ and ϕ_{L2}.

C. Communications Models

Queueing theory provides an analytical model for the relationship between bandwidth and delay in stochastic systems. In addition to network analyses, queueing theory has been successfully applied to modeling multiprocessor server performance [12]. In our approach, N processor cores produce transactions requesting access to the finite bandwidth main memory bus, while the shared memory controller arbitrates between the requesting cores. If the bus is busy, transactions must wait in a FIFO queue until they are granted access to the bus. Each core has a dedicated queue and channel to the memory controller. The average time spent in a Markovian M/M/1 queue [13] is:

$$\bar{t} \quad = \quad \frac{1}{\mu - \lambda} \quad (11)$$

where μ is the average service rate of the queue, and λ is the average rate at which transactions arrive, going into the queue.

TABLE I
PROCESSOR CONFIGURATIONS

Point	Configuration	Integer ALU/Mult	Float Point ALU/Mult
A	half ALU	2/1	1/1
B	baseline (4-way)	4/1	2/1
C	double ALU	8/2	4/2
D	quad ALU + (16-way)	8/4	8/4

With $\lambda = \alpha_0 N$, the main memory latency D_2 as a function of N is:

$$D_2 \quad = \quad \alpha_1 + \bar{t} = \alpha_1 + \frac{1}{\mu - \alpha_0 N} \quad (12)$$

where $\alpha_0 = G_2/CPI$ is a constant scaling the rate of arrival, and α_1 is a constant defining the minimum latency in cycles of a memory access.

D. Area Parameters

Decomposing the total area A_{tot} into its fundamental components, we form the fixed area constraint:

$$A_{tot} \quad = \quad N(A_P + A_{L2}) + A_{fix} \quad (13)$$

where N is the number of cores in the CMP, A_P is area of a processor core, A_{L2} are the area of the L2 cache, and A_{fix} accounts for the fixed area functions (I/O, memory controller, test and debug circuitry, etc.) We constrain our optimization to find the solution that satisfies this fixed area constraint. The L1 cache could be explicitly modeled, instead of aggregated into A_P. However, we chose not to because the typical L1 cache size is 16 to 64 times smaller than an L2 cache, and thus is a much smaller variable in the overall optimization.

V. PARAMETER ESTIMATION

In this section, we estimate the constant parameters for each of the lower-level models using empirical data from speech recognition applications. The theory lines are fit to the data by choosing parameters values that minimize the mean squared error (MSE) between the collected data and the theory line.

A. Processor Models

In order to estimate the parameters β and ϕ_P for (3), we need two pieces of information: processor performance for various processor microarchitectural configurations and the corresponding die area for those processor configurations. We use the SimpleScalar ISS [14] to estimate the processor performance for each configuration while running the Sphinx 3.0 speech recognizer [15], following the approach of [16].

The baseline processor is a single core of an Intel Core2Duo. Important SimpleScalar values for the baseline processor configuration include: 64KB eight-way associative IL1 and DL1 caches, a 2MB four-way L2 cache, an out-of-order core, four instruction fetch/issue/decode per cycle, four integer ALUs, single integer multiply, two floating-point ALUs, and a single floating-point multiplier. To measure the processor performance for different chip areas, we evaluated the configurations summarized in Table I.

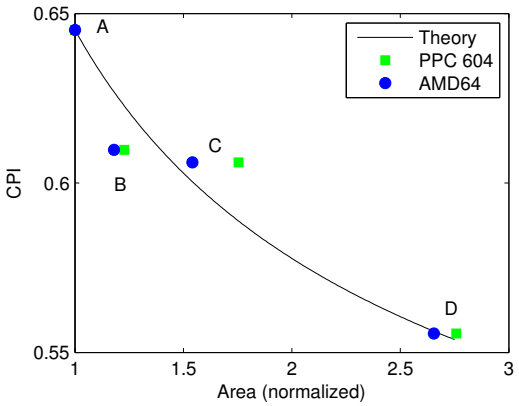

Fig. 3. Processor performance (CPI) as a function of processor area. Labeled processor configurations are defined Table I.

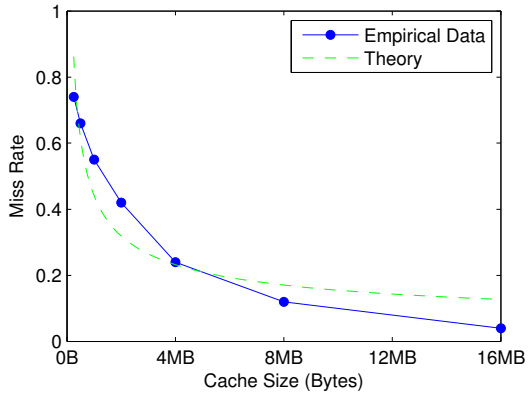

Fig. 4. L2 cache miss rate as a function of cache size

We estimated the corresponding superscalar processor microarchitecture areas using labeled die photos of comparable superscalar microarchitectures, the AMD64 dual core CMP [17] and the PPC 604 single core processor [18]. After measuring the area of the labeled functional units, including integer and floating point execution units, fetch and load/store logic, we linearly scale the area in terms of the number of functional units or fetch/issue width. The results from combining the CPI results with the estimated processor areas are plotted in Fig. 3. The theory line is plotted with the following minimum MSE parameters: $\phi_P = 0.415$ and $\beta = 117.8$.

B. Cache Memory Models

Like the processor model parameters, we estimate the cache memory model parameters κ and ϕ_{L2} for (7) using SimpleScalar running Sphinx 3.0. For these experiments we start with the baseline configuration and vary the L2 cache size to measure the effect on the miss rate. In Fig. 4 we plot the miss rate for the L2 cache with sizes ranging from 256KB to 16MB. The minimum MSE parameters are: $\phi_{L2} = 0.02$ and $\kappa = 430$, resulting in the theory line plotted in the figure.

C. Communications Models

For the M/M/1 queue model of off-chip memory delay (12), we have to estimate the queue service rate μ, as well as two free parameters: α_0 and α_1. In order to estimate these values, we assume a 2.4GHz CMP clock frequency and a DDR2-800 main memory interface [19]. The DDR2 memory interface supports seamless burst read or write accesses back-to-back. However, the turn around time between read and write accesses is 4 cycles (R to W) and 10 cycles (W to R). Assuming balanced reads and writes, the average R/W turnaround time is 7 cycles. The additional overhead time for DRAM refresh is negligible (less than 1% of the time). If we are able to make back-to-back accesses 40% of the time and the DDR2 transfer rate is three times slower than the CMP clock frequency, then the average service rate in cycles is: $\mu = (0.4 \times 3 + 0.6 \times 3 \times 7)^{-1} = 0.0725$.

The average rate at which transactions arrive at the queue is λ. It is the fraction of instructions accessing main memory G_2 divided by the average cycles per instruction (CPI) of the processor multiplied by the number of processor cores in the CMP N, or: $\lambda = N\alpha_0 = N \times G_2/CPI$. Now $CPI = G_0D_0 + G_1D_1 + G_2D_2$ and $G_2 = (1 - G_0)(\kappa A_{L2}^{-\frac{1}{2}} + \phi_{L2})$, containing the variables that we are trying to solve for in the optimization. This apparent recursive relationship is solved using an iterative approach analogous to the Expectation Maximization (EM) algorithm. Given an initial guess for CPI and G_2, we find the optimal architecture. Then we use the optimal architectural parameters to re-estimate CPI and G_2. After a few iterations, the value of α_0 converges. (Within two iterations it converges to within less than 0.5% of its final value).

The parameter α_1 accounts for the latency of the memory access. For a recent CMP, the Cell processor [20], the minimum latency to write to memory is 290 cycles, while the minimum read latency is 580 cycles. Without differentiating between the read and write latency and a 25% lower clock frequency than the Cell processor, we use a minimum main memory latency of $\alpha_1 = 325$ cycles.

D. Application Parameters

The parameter F_j in (1) accounts for the parallelism in the application. Using only two algorithm fractions ($K = 2$), then F_0 is the serial fraction of the algorithm and F_1 is the parallel fraction of the algorithm. With a symmetric architecture, the F_j parameters are estimated independent of the other microarchitectural parameters. First, we run the algorithm multiple times using a range (1 to 50) of traditional parallel processors (1 core per chip). For each run, we observe the number of processors used and the time to decode the full speech dataset. The parallel speedup is the ratio of the single processor execution time to the N processor execution time:

$$S = \frac{t_{1proc}}{t_{Nproc}} = \frac{1}{\left(F_0 + \frac{F_1}{N}\right)} \tag{14}$$

N, t_{1proc}, and t_{Nproc} are empirically measured, and since $F_0 = 1 - F_1$, we can solve for F_1, by finding the value

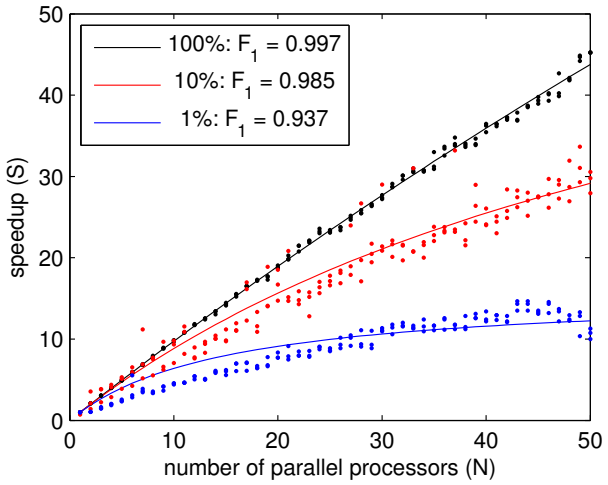

Fig. 5. Speedup vs. number of processors (N) for estimating parallel fraction of the application F_1. Each curve represents a different quantity of speech being decoded.

that will minimize the MSE between the empirically measured execution times and the theory line.

For our CMP optimization example, we estimated F_1 on a speech digit recognition task using the TIDIGITS speech corpus. We used the HTK speech recognizer [21] to decode each spoken string of numerical digits, a task that involved evaluating Gaussian Mixture Models followed by Hidden Markov Models. We created three different datasets, differing by the amount of evaluation data used (100%, 10%, 1% respectively). The number of parallel processors N and the corresponding speedup S for each trial and each data set size is plotted in Fig. 5. F_1 is estimated for each dataset by fitting (14) in a least squares sense to the data. The fitted curve and estimated value of F_1 for each dataset is shown in the legend of Fig. 5. As the dataset gets smaller, the time spent in the serial fraction of the algorithm increases relative to the total execution time. This is intuitive because tasks such as initializing the GMM and HMM model parameters must be done and take the same amount of time regardless of the decoded dataset size. Notice that F_1 is close to 1.0, and therefore decoding large speech datasets has a significant level of parallelism.

The final parameter to estimate is the fraction of hits at each level of the memory hierarchy G_{ij}. We estimated the number of memory accesses by counting the number of instructions in the assembly code that contain a memory reference. We compiled the GMM computation code from HTK using `gcc` for the x86 microarchitecture with the assembly flag set. Using the resulting assembly code, we manually counted the memory references, taking into account the number of loop iterations and code that conditionally was or was not executed. From [16], we have an L1 D-cache hit rate of 0.975 for a 128KB L1 D-cache and assume a 1.0 hit rate in the L1 instruction cache. Using a memory access frequency of 0.55 from Table II, the fraction of instructions whose data is found only in the register file or L1 cache is: $G_0 = (1-0.55)+0.975\times 0.55 = 0.9862$. The other 0.0138 fraction of instructions retrieve their data from the L2 cache or beyond in main memory.

TABLE II
MEMORY ACCESS FREQUENCY

Configuration	1 Mixture	8 Mixtures
# Instructions	437	3070
# Memory Accesses	237	1704
Memory Access Frequency	0.5423	0.5550

VI. OPTIMIZATION

Combining the high-level objective function (1), with the lower-level models of processors (4), cache memory (8–10), and shared access communication (12), we form the high-level cost function for our architectural model:

$$\begin{aligned} J_D &= \left(F_0+\frac{F_1}{N}\right)\Big[G_0(\beta A_P^{-\frac{1}{2}}+\phi_P)+ \\ &\quad (1-G_0)(1-\kappa A_{L2}^{-\frac{1}{2}}-\phi_{L2})D_1+ \\ &\quad (1-G_0)(\kappa A_{L2}^{-\frac{1}{2}}+\phi_{L2})\left(\alpha_1+\frac{1}{\mu-\alpha_0 N}\right)\Big] \end{aligned} \tag{15}$$

Combining the cost function (15) with the area constraint (13), we form the Lagrangian for constrained optimization:

$$\begin{aligned} L(A_P, A_{L2}, N, \lambda) &= J_D + \\ &\quad \lambda[N(A_P+A_{L2})+A_{fix}-A_{tot}] \end{aligned} \tag{16}$$

To find the optimal architecture, we differentiate the Lagrangian with respect to the four variables[2], resulting in a system of non-linear equations with four equations and four unknowns $(A_P, A_{L2}, N, \lambda)$. The closed form solution to this particular system is intractable. As a result, we use Newton's method [22] to find the optimal values of the four unknowns. With these optimal values, the cost function J_D is at a minimum[3] and corresponds to the optimal CMP architecture. Fig. 6 depicts the cost surface J_D graphed for two of the variables A_P and A_{L2}. The third variable N, takes the value that satisfies the area constraint (13). The optimal architecture is designated by the black dot. Similar graphs exist for A_P vs. N and A_{L2} vs. N.

The update rule for the variables $\mathbb{X}$ for each iteration k of Newton's method is:

$$\mathbb{X}(k+1) = \mathbb{X}(k) - (\mathbb{F}/\mathbb{J})^T \tag{17}$$

where $\mathbb{X} = [A_P, A_{L2}, N, \lambda]^T$ and $\mathbb{F} = \left[\frac{\partial L}{\partial A_P}, \frac{\partial L}{\partial A_{L2}}, \frac{\partial L}{\partial N}, \frac{\partial L}{\partial \lambda}\right]$ and $\mathbb{J}$ is the Jacobian of $\mathbb{F}$.

[2]This differentiation is automatically performed using symbolic math software such as Maple or Sage.

[3]In this example, J_D is convex, resulting in a global minimum. If J_D is non-convex, simulated annealing or stochastic gradient descent can be used to find the global minimum.

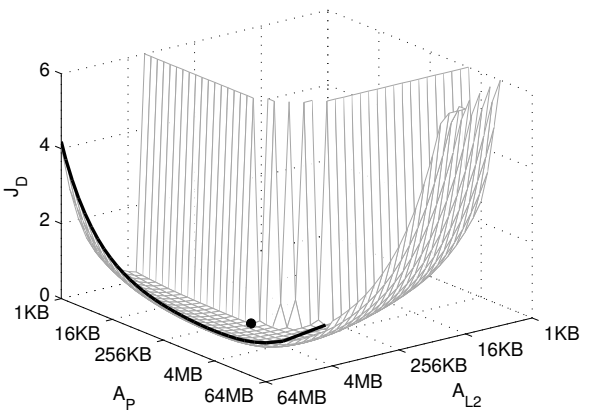

Fig. 6. Constrained cost surface J_D with the optimal architecture designated by the black dot. Units of J_D are cycles (per instruction). Areas are given in memory equivalent units.

VII. Results

For the given parameters, the optimal symmetric CMP architecture is: $N = 23.04 \approx 23$ processor cores, $A_P = 2^{18.87}$ which is approximately the area of a $512KB$ cache memory, and $A_{L2} = 2^{21.4} \approx 3MB$. We can see two clear effects. First, since the parallel fraction of the algorithm is close to 1.0, then the optimal architecture consists of many cores of smaller size. Second, the L2 cache is roughly six times larger than the processor area. This is due to the high L2 miss rate and the role of the L2 cache in reducing the number of expensive off-chip memory accesses.

From Fig. 6, we can make several observations about the cost surface. First, the cost surface resembles an "L" shaped canyon, with a sharp (nearly vertical) back wall and a front wall that rises to the left and right. The back wall occurs at small values of A_P and A_{L2}, and thus large values of N. With large N, the shared memory interface becomes a bottleneck, rapidly increasing the delay due to congestion on the communication interface. The front wall occurs for large values of A_P or A_{L2}, corresponding to small values of N. In this region, the cost primarily increases because the architectures could utilize more parallelism to increase performance, but do not. The two sides of the front wall increase as either A_P or A_{L2} become unbalanced with respect to the other. The front wall is bounded by the architectures that exceed the total area A_{tot} as indicated by the thick black line.

VIII. Conclusion

In summary, we have presented three significant advancements. First, we present a CMP objective function with models of processors, cache memory, and communication in an area-constrained, global optimization process. Second, we apply the objective function to the task of optimizing a symmetric CMP for a specific application, automatic speech recognition, and demonstrate methods for estimating the parameters of the underlying models. Third, we present an automated approach to solving the objective function for the optimal architecture.

We have assumed parallel decoding on a symmetric architecture, where each core is running an identical instance of the speech recognizer. Alternatively, we could assume an asymmetric case and partition the recognizer algorithm over different cores. This would yield a different optimal architecture. Our current research is focused on extending our objective function to asymmetric architectures.

References

[1] M. Hill and M. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[2] D. H. Woo and H. Lee, "Extending Amdahl's law for energy-efficient computing in the many-core era," *Computer*, vol. 41, no. 12, pp. 24 – 31, Dec 2008.

[3] T. Oh, H. Lee, K. Lee, and S. Cho, "An Analytical Model to Study Optimal Area Breakdown between Cores and Caches in a Chip Multiprocessor," in *IEEE Computer Society Annual Symposium on VLSI*. IEEE Computer Society, 2009, pp. 181–186.

[4] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *International Symposium on High-Performance Computer Architecture (HPCA)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 340–351.

[5] X. Chen and T. Aamodt, "A first-order fine-grained multithreaded throughput model," in *International Symposium on High Performance Computer Architecture, (HPCA)*, 2009, pp. 329–340.

[6] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron, "CMP design space exploration subject to physical constraints," *International Symposium on High-Performance Computer Architecture*, pp. 15 – 26, Feb 2006.

[7] M. Monchiero, R. Canal, and A. González, "Design space exploration for multicore architectures: a power/performance/thermal view," *International Conference on Supercomputing (ICS)*, Jun 2006.

[8] A. Cassidy and A. G. Andreou, "Analytical methods for the design and optimization of chip-multiprocessor architectures," *43rd Annual Conference on Information Sciences and Systems (CISS 2009)*, pp. 482–487, Mar 2009.

[9] S. Borkar, "Thousand core chips: a technology perspective," *Proceedings of the 44th annual Design Automation Conference (DAC '07)*, Jun 2007.

[10] S. Przybylski, M. Horowitz, and J. Hennessy, "Characteristics of performance-optimal multi-level cache hierarchies," *International Symposium on Computer Architecture (ISCA)*, pp. 114–121, Apr 1989.

[11] A. Hartstein, V. Srinivasan, T. Puzak, and P. Emma, "On the Nature of Cache Miss Behavior: Is It $\sqrt{2}$?" *The Journal of Instruction-Level Parallelism*, vol. 10, 2008.

[12] R. Matick, T. Heller, and M. Ignatowski, "Analytical analysis of finite cache penalty and cycles per instruction of a multiprocessor memory hierarchy using miss rates and queuing theory," *Ibm J Res Dev*, vol. 45, no. 6, pp. 819–842, Jun 2001.

[13] D. Gross and C. M. Harris, *Fundamentals of queueing theory*, 3rd ed. John Wiley & Sons, Inc., 1998.

[14] T. Austin, E. Larson, and D. Ernst, "Simplescalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59 – 67, Feb 2002.

[15] P. Placeway, S. Chen, M. Eskenazi, U. Jain, V. Parikh, B. Raj, M. Ravishankar, R. Rosenfeld, K. Seymore, M. Siegler *et al.*, "The 1996 Hub-4 Sphinx-3 System," in *Proc. DARPA Speech recognition workshop*, 1997, pp. 85–89.

[16] K. Yu and R. A. Rutenbar, "Profiling large-vocabulary continuous speech recognition on embedded devices: A hardware resource sensitivity analysis," *Interspeech*, pp. 1923 – 1926, Sep 2009.

[17] "AMD website," *http://www.amd.com/*, 2009.

[18] L. Gwennap, "PPC 604 Powers Past Pentium," *Microprocessor Report*, pp. 5–8, 1994.

[19] "DDR2 SDRAM Device Operating & Timing Diagram," *Samsung Datasheet*, 2007.

[20] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: built for speed," *IEEE MICRO*, vol. 26, no. 3, pp. 10 – 23, May 2006.

[21] S. Young, P. Woodland, and W. Byrne, "HTK-Hidden Markov Model Toolkit," *Cambridge University Engineering Department*, 1999.

[22] R. Burden and J. Faires, *Numerical Analysis*, 5th ed. PWS Publishing Company, 1993.