Efficient Cache Exploration Method for a Tiled Chip Multiprocessor

Aparna Mandke Dani
Indian Institute of Science
aparnam@csa.iisc.ernet.in

Y. N. Srikant
Indian Institute of Science
srikant@csa.iisc.ernet.in

Bharadwaj Amrutur
Indian Institute of Science
amrutur@ece.iisc.ernet.in

***Abstract*—Past studies use deterministic models to evaluate optimal cache configuration or to explore its design space. However, with the increasing number of components present on a chip multiprocessor (CMP), deterministic approaches do not scale well. Hence, we apply probabilistic genetic algorithms (GA) to determine a near-optimal cache configuration for a sixteen tiled CMP. We propose and implement a faster trace based approach to estimate fitness of a chromosome. It shows up-to 218x simulation speedup over the cycle-accurate architectural simulation. Our methodology can be applied to solve other cache optimization problems such as design space exploration of cache and its partitioning among applications/virtual machines.**

***Keywords*-genetic algorithms, performance evaluation, chip multiprocessors**

I. Introduction and Related Work

The number of cores and on-chip cache size have been increasing on CMPs due to advances in technology. As a result, leakage power consumed by caches has become a major contributor to the power dissipated in the memory subsystem [1]. Dissipation of the leakage power can be reduced by estimating cache requirement of an application and then switching off the over-allocated cache.

Past studies that determine optimal cache configuration use offline heuristic based search methods [2]. Sugumar et al. [3] use binomial trees to simulate a group of cache configurations in a single simulation for a uniprocessor platform. Avissar et al. [4] formulate a problem of allocating variables to a scratch pad memory in an embedded systems as a 0/1 integer linear program. It is possible to use such deterministic algorithms in the case of embedded systems as those are simpler and applications that are executed on them are also simpler than multithreaded applications executing on a CMP. Dinero [5] proposed a trace based simulator which predicts cache misses for a given cache configuration from a memory address trace. However, it is applicable for uniprocessor systems only. In the case of a CMP, predicting interleaving of accesses made by various threads is important. Tam et al. [6] collect accesses made to the L2 cache using hardware performance counters and generate cache miss rate curve using Mattson's stack algorithm [7] for a dual core scenario. CMPSim [8] achieves simulation speedup by using binary instrumentation technique and show its applicability up-to four threads scenario on a CMP.

However, due to presence of concurrently executing threads and increased number of components, such deterministic algorithms cannot scale for a CMP. Hence, we apply probabilistic GA to determine the near optimal cache configuration for a tiled CMP. Diaz et al. [9] used GA to optimize cache for the first time. They use it to partition cache ways among two applications executing on a simultaneous multi-threading platform. However, they use cycle-accurate architectural simulation to find fitness of a chromosome which is very slow and hence they use population size of 30 chromosomes only. On the contrary, we propose a trace based method to estimate fitness of a chromosome and demonstrate its scalability for a sixteen core and multi-threaded workloads scenario. Unlike our method, prior studies have used trace based approaches mainly for a single threaded workloads [5]. Following are our major contributions in this paper:

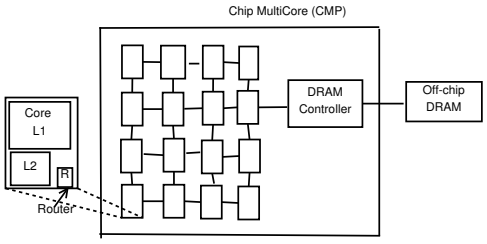


Fig. 1. Tile architecture used for study

1) Application of probabilistic GA method to estimate the near optimal cache configuration for a larger scalable CMP: This is achieved by using a novel trace based approach, which shows up-to 218x speedup over the cycle-accurate architectural simulation.
2) Proposal of a trace based cache simulator for a shared cache on a CMP - We modify Dinero [5] to predict cache misses incurred by a shared cache when accessed by simultaneously executing threads on a CMP. Our results show that Dinero can be used for simulating a shared non-inclusive cache on a CMP.

Our paper is organized as follows. We give background of our research problem in section II, describe our GA formulation in section III. Section IV describes the experimental setup. Results and conclusions are explained in section V and section VI, respectively.

978-1-4673-2371-0/12/$31.00 ©2012 IEEE

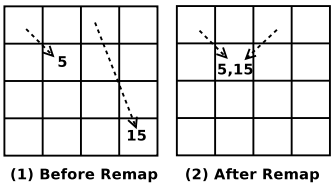
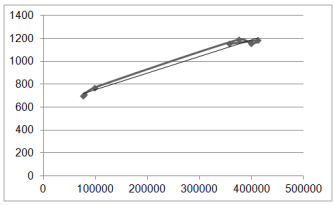
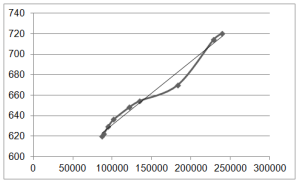


Fig. 2. The Remap Policy



(a) mpegenc: Y = 0.00136*X + 633.34



(b) x264: Y = 0.00066*X + 562.0644

Fig. 3. plots power dissipated in DRAM (in mW) on the Y axis against the number of accesses on the X axis. We use model based on the linear interpolation to estimate power dissipated in DRAM.

II. Background

We consider a sixteen tiled CMP architecture as it is advocated as a scalable CMP (Fig. 1). Each tile contains a core, a split instruction and data L1 cache. Each tile also contains a slice of L2 cache. The L2 cache is distributed and shared by all cores. To maintain cache coherence between private L1 caches, a directory is present in each tile. These tiles are inter-connected via 2D-mesh switched network (NoC) and per-tile router.

Large caches are implemented using multiple banks that are inter-connected with a on-chip network. Such a cache offers non-uniform access latency to different cores present on a CMP. Hence, it is called Non-Uniform Cache Architecture (NUCA) [10]. Kim et al. [10] proposed static NUCA (SNUCA) access policy. In SNUCA, the predetermined bits of a memory address decide the tile where data is cached, referred to as home location of that address. However, such a cache suffers from increased access latency due to its distributed nature.

We simulated various applications from Splash2 [11] and Parsec [12] benchmark suite. These experiments show that due large caches present on the modern processors (8MB-16MB), cache requirement of majority of applications is lesser than the available cache. Hence, we proposed the remap policy to reduce the leakage power consumed by the caches [1]. In the remap policy, farther L2 slices are mapped on-to nearer L2 slices and farther L2 slices are switched-off. An L2 slice includes all L2 banks in a tile. Apart from reducing leakage power consumption in NUCA caches, the remap policy also improves execution time in some applications.

Fig. 2 shows working of the remap policy. The L2 slice in tile 15 is mapped on-to the L2 slice in tile 5. In this case, L2 slice in tile 5 caches data of both the 5^{th} and 15^{th} L2 slices. On an L1 miss, the remap configuration maintained in each L1 controller, is looked up and if the home location is L2 slice in tile 15 then a request is sent to the L2 slice in tile 5. In the remap policy, a tile on-to which 15^{th} L2 slice is mapped, is also important. Time spent in interconnect traversal increases if a tile is mapped to a farther slice.

In this study, we want to determine the near-optimal remap configuration (NORC). Energy savings obtained with the NORC will be used to compare energy savings obtained with the dynamic remap configuration policy [1]. The NORC should allocate just the required number of L2 slices. The L2 slices that are to be switched off, should be mapped on-to allocated L2 slices such that the number of cache misses do not increase significantly. The leakage power dissipated in the cache is proportional to size of the powered-on cache. If lesser than the required cache is allocated, then the number of off-chip DRAM accesses increase, degrading execution time of an application. On the contrary, the leakage power consumption of cache increases on allocating more than the required cache. Hence, we use energy-delay product (EDP) as a performance metric which indicates a trade-off between energy consumption of an application and its execution time (delay). EDP for an application is calculated as $EDP = Energy * ExecutionTime$.

Clearly, we can exhaustively simulate all possible remap configurations and determine the maximum EDP savings that can be obtained using the remap policy. However, architectural simulation is extremely slow and this method will not scale with increasing number of tiles on a CMP. Hence, we solve this problem using probabilistic GA and propose a faster method to evaluate fitness of each remap configuration.

III. Problem Formulation

We consider energy consumed by various memory subsystem components such as off-chip memory, interconnect and on-chip cache while determining EDP of an application. To estimate the execution time of an application, we predict cache misses and time spent in transit for a given remap configuration using a trace based approach. The meaning of various terms used in our problem formulation is described in Table I.

A. Estimation of Energy Consumption

E_cache**:** Leakage power is proportional to the number of allocated L2 slices. Leakage power per L2 slice is obtained using CACTI [13]. The total dynamic energy dissipated in the L2 slices remains nearly same for all remap configurations. Hence, we ignore dynamic energy consumption in the objective function.

$E\_interconnect$**:** Interconnect power does not vary significantly if execution time of an application for a remap configuration is comparable to the reference. Hence, we ignore this term as well in the objective function.

E_DRAM**:** Power consumed in the off-chip DRAM depends on the number of memory accesses, row management policy and address correlation between the consecutive

Parameter	Description	Method of Determination
EDP	Energy-Delay Product	Eq. 1
E_cache	Energy consumed by the cache	CACTI[13]
$E\_interconnect$	Energy consumed by interconnect	-
E_DRAM	Energy consumed by off-chip DRAM	Statistical model of data obtained using DRAMSim[14]
$Time$	Time taken by an application to complete its execution	-
C_{total}	Total time spent in servicing all L1 misses	Eq. 6
C_{remap}	Total time spent in servicing all L1 misses for a given remap configuration	Eq. 7
K	# of threads used by an application for execution	Specification
N	# of tiles/L2 slices on a CMP	Specification
S	# of L2 slices allocated to an application	GA (Section III-C)
η_{ij}	# of accesses between L1 cache in tile i and L2 slice in tile j	Profiled information
L_{ij}	Network access latency between L1 cache in tile i and L2 slice in tile j	Floorplanning, Intacte[15]
α_{ij}	# of cache misses incurred by L2 slice in tile j due to traffic generated by thread executing on a core in tile i	Dinero (Section III-C)
p_j	Penalty to service a single cache miss in L2 slice in tile j. This penalty includes latency of a memory access and network latency between L2 cache in tile j and the memory controller.	Intacte, an average of DRAM latencies obtained using DRAMSim
δ	remap function, $\delta(j) = i$ refers to L2 slice in the tile j is mapped to L2 slice in the tile i. $\delta(i) = i$ refers to L2 slice from tile i is mapped to itself.	GA (Section III-C)
K_c	Constant decides priority given to the cache miss penalty	Empirically
K_t	Constant decides priority given to the transit time penalty	Empirically

TABLE I
SHOWS MEANING OF THE VARIOUS TERMS USED IN GA FORMULATION. THE LAST COLUMN DESCRIBES HOW THESE TERMS ARE OBTAINED.

accesses. We simulated applications by varying the number of powered-on L2 slices and obtained power consumed by the off-chip DRAM. Our experiments show that the power consumed by the memory remains almost same if the number of DRAM accesses are comparable to the reference. We use the closed row management policy, hence, correlation between the consecutive memory accesses can be ignored. We use an approximate memory power consumption model based on the linear interpolation of empirical data. Fig. 3 shows the models used to estimate power consumed by the offchip memory for various remap configurations. Due to insufficient space, we have shown models only for two applications. If significantly higher memory accesses are predicted for a remap configuration than the reference, then its EDP will be higher due to higher power dissipated in DRAM. If the number of memory accesses are low due to over-allocation of L2 slices, then EDP will be higher due to excessive leakage power dissipated in the on-chip cache. In both these cases, the solution will be discarded by the GA. This ensures that the GA allocates the right number of the L2 slices. With these assumptions, EDP is simplified as:

$$EDP \approx ((LeakagePwrPerL2Slice * \#OfAllocatedL2Slices + memPwr) * Time) * Time \quad (1)$$

B. Estimation of Time

$Time$ component of Eq. 1 is expressed in terms of the processor clock cycles. It is proportional to the time spent in servicing L1 cache misses and traversing NoC.

Estimation of Transit Time : Total time spent by a thread i in transit is equal to the traffic generated by the thread *i* between L1 cache in tile *i* and all L2 slices. Formally,

$$\tau_i = \sum_{0 \le j < N} \eta_{ij} L_{ij} \quad (2)$$

Hence, the total time spent in transit by all the threads is,

$$\tau_= \sum_{0 \le i < K} \tau_i = \sum_{0 \le i < K} \sum_{0 \le j < N} \eta_{ij} L_{ij} = \sum_{0 \le j < N} \sum_{0 \le i < K} \eta_{ij} L_{ij} \quad (3)$$

Since all the threads execute concurrently, time spent in transit by these threads overlaps. The sum of their timings denotes the maximum time an application might spent in transit. The trace between L1 caches and L2 slices also includes traffic caused due to coherence.

Estimation of Cache Miss Penalty : Execution time of an application is greatly affected by the cache miss ratio, especially, in the last level cache (LLC) of the cache hierarchy. Therefore, the remap configuration should be chosen such that the total number of L2 cache misses should not increase significantly. The cache miss cost c_j of an L2 slice j is proportional to the penalty p_j incurred on a miss in the L2 slice in tile j and the number of misses generated in the L2 slice j by all threads.

$$c_j = \sum_{0 \le i < K} p_j . \alpha_{ij} \quad (4)$$

Total cache miss penalty, C_{Miss} incurred by all L2 slices is,

$$C_{Miss} = \sum_{0 \le j < N} c_j = \sum_{0 \le j < N} \sum_{0 \le i < K} p_i . \alpha_{ij} \quad (5)$$

Hence, the total cost incurred by misses in all L1 caches is the sum of costs given by Eq. (3) and Eq. (5).

$$C_{total} = \sum_{0 \le j < N} \sum_{0 \le i < K} \eta_{ij} L_{ij} + \sum_{0 \le j < N} \sum_{0 \le i < K} p_j . \alpha_{ij} \quad (6)$$

GA chooses the remap configuration such that C_{total} is minimized. We give more weightage to the cache miss penalty than the transit time penalty since latency of one miss in the LLC is much higher than its transit time penalty. We

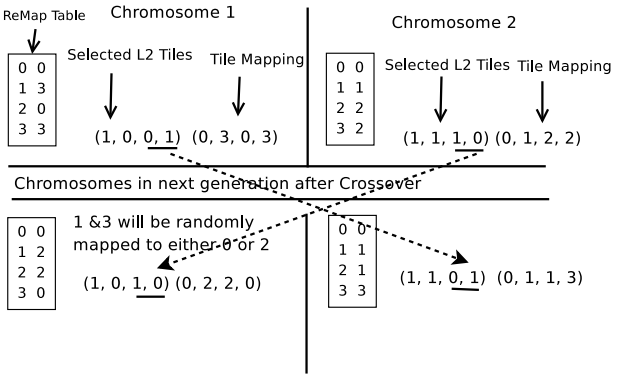


Fig. 4. The remap configuration is encoded as a chromosome. Fig. shows the crossover operation.

Name	Domain	Description, WSS(L/M/S)
Alpbench Benchmark [16]		
mpegenc mpegdec	Media	Encodes/decodes 15 Frames of size 640x336, M
Splash2 Benchmark [11]		
fft	Signal processing	FFT on 1M points, M
radix	General	1M keys, M
raytrace	Graphics	car input file, M
cholesky	HPC	sparse matrix factorization on tk29, L
water_spatial	HPC	512 water molecules, M
water_nsquared	HPC	512 water molecules, M
barnes	HPC	Barnes-Hut method on 16K bodies, M
ocean (continuous)	HPC	512x512 grid points, L
PARSEC Benchmark [12]		
blackscholes	Financial	SimLarge i/p, S
swaptions	Financial	SimLarge i/p, M
fluidanimate	Animation	SimMedium i/p, L
x.264	Media	SimLarge i/p, M

TABLE II
SHOWS APPLICATIONS USED FOR STUDY AND THEIR WSS INFORMATION (L:LARGE, M:MEDIUM, S:SMALL)

empirically found that $K_t = 1$ and $K_c = 8$ remap switched-off L2 slices more uniformly[1]. We use these values of K_c and K_t in all our experiments. Hence, for a given remap table, the $Time$ component is,

$$C_{remap} = K_t \sum_{0 \le j < N} \sum_{0 \le i < K} \eta_{ij} L_{i\delta(j)} + K_c \sum_{0 \le j < S} \sum_{0 \le i < K} \alpha_{ij} p_j \quad (7)$$

To estimate the number of L2 cache misses for a given remap table, we modify a trace-driven Dinero cache simulator [5] by replicating cache hierarchy for the multiple L2 slices with the shared off-chip memory. We simulate an application with all L2 slices and capture traffic generated between all L1 and L2 caches in a trace. The modified Dinero simulator requires the trace and a remap configuration as an input. It estimates the number of cache misses incurred by each L2 slice for a given remap configuration. We consider a CMP with non-inclusive shared L2 cache in which, conflict misses in L2 do not cause eviction in L1 cache. And most of the accesses are L1 hits. Hence, this method gives reasonably accurate estimation of L2 cache misses. The same trace is used to determine the transit time cost.

C. Genetic Algorithm Formulation (GA)

The GAs are a class of randomized search algorithms useful in function optimization. A possible solution of the problem is encoded as a fixed size binary array, called chromosome. We use a chromosome to represent one remap configuration. As shown in Fig. 4, the remap configuration is represented as 2 strings. 1^{st} binary string gives the L2 slices allocated to an application and 2^{nd} gives the mapping of unallocated slices e.g. a chromosome "(1,0,0,1)(0,3,0,3)" indicates that 0^{th} and 3^{rd} L2 slices are allocated. 1^{st} and 2^{nd} L2 slices are mapped on-to the 3^{rd} and 0^{th} slice, respectively. The details of our GA formulation are as follows:

- EDP in Eq. 1 gives the fitness value of a chromosome.
- The population size is 100 chromosomes.
- The crossover is performed at a randomly selected single point between two chromosomes with probability of 0.65. Crossover is done only on the first binary string which represents the allocated L2 slices. The unallocated L2 slices are randomly mapped on-to the allocated L2 slices. Each remapped slice is mutated with another allocated L2 slice with probability of 0.015. We have chosen mutation and crossover probabilities within the standard range [9].
- Generally, in the case of GA, higher fitness value of a chromosome indicates better suitability. But in our problem, lower EDP value indicates better remap configuration. Hence, to convert this minimization problem into a maximization problem, fitness value of all chromosomes is subtracted from the largest fitness value of the population.
- We use the roulette wheel selection method. The search algorithm is terminated after 80 generations. The fittest solution is used for further evaluation. Our GA implementation stabilizes after 20-30 generations for all applications.

IV. Experimental Configuration

We evaluate multi-threaded workloads with one-to-one mapping between threads and cores (Table II). We have carefully chosen applications from Splash2 [11], Alpbench [16] and Parsec [12] benchmark suites such that they cover programs with different characteristics and domains ranging from high performance computing, financial domain, animation, media and signal processing [2]. We have skipped initial serial portion and simulate only parallel section in all the tests. We test all workloads with 16 threads and simulate 1 billion instructions.

Table III shows the system configuration used in our experiments. We use SESC [17] to simulate a core, Ruby component from GEMS [18] to simulate the cache hierarchy and interconnects. DRAMSim [14] is used to model the

[1]We are unable to give sensitivity data for various K_t and K_c values due to space limitation.

[2]Rest of the PARSEC benchmarks either use OpenMP APIs or libraries which are not supported by SESC compiler, hence cannot be compiled using SESC compiler.

Core	out-of-order execution, 3GHz, issue/fetch/retire width of 4
L1 Cache	32KB, 2-way, 64B, access latency of 2 cycles (estimated using CACTI), private, cache coherence using MOESI protocol
L2 Cache	512KB/tile, 16 way, 64B line size, 3 cy. latency (estimated using CACTI), noninclusive, shared and distributed across all tiles
Interconnect	16 bits flit size, 4x4 2D MESH, deterministic routing, 4 virtual channels/port, credit based flow control, router queues with length of 10 buffers
DRAM	4GB, DDR2, 667MHz freq, 2 channels of 8B in width, 8 banks 16K rows, 1K columns, close page row management

TABLE III
SYSTEM CONFIGURATION USED IN EXPERIMENTS

offchip DRAM. Intacte [15] is used to estimate low level parameters of the interconnect such as the number of repeaters, wire length, and power consumed by the interconnect. Power consumed by the cache components and their area is estimated using CACTI 6.0 [13].

V. RESULTS

We evaluate accuracy of the method used to estimate cache misses and execution time. We demonstrate that the NORC obtained using our GA formulation gives comparable performance to that determined using exhaustive search method.

A. Accuracy of Cache Misses Estimation Method

As mentioned earlier, to estimate the number of cache misses incurred for a remap configuration, we have modified Dinero cache simulator [5]. It takes a trace between L1 and L2 caches and a remap configuration as an input and estimates the number of cache misses incurred by each online L2 slice. To check accuracy of this method, we modified GA algorithm to obtain the remap configuration which allocates 2, 4, 6 .... L2 slices. The remap configurations obtained using this method are simulated. The cache misses are also estimated using our method for those remap configurations. Table IV gives correlation between the two values. All the applications show very good correlation of an average of 97%. This is because, typically CMPs have a shared non-inclusive LLCs. Hence, eviction of a cache line from an L2 cache does not cause eviction from L1 cache(s). As majority of accesses are L1 hits, interleaving of cache accesses made by different threads does not show significant increase in cache misses. In the case of applications with large working set size, this might cause some inaccuracies. Despite this, ocean and cholesky show 99% correlation between the cache misses predicted using the two methods.

B. Accuracy of execution time estimation method

We model total time spent in transit and in cache misses using Eq. 7. To evaluate accuracy of this method, we plot execution time obtained using Eq. 7 and that obtained using simulation for various remap configurations (Fig. 5). These plots should not be compared by their absolute values since the execution time depends on the memory level parallelism. Time obtained using Eq. 7 estimates the total time spent in transit and cache misses. Execution time is much lower than this due to memory level parallelism and out of order execution on a core. Hence, a trend in values (on varying the number of allocated L2 slices) is more important than their absolute values. Fig 5 clearly shows a good correlation between the two values. We observed similar trend for other applications as well. This shows that our method can be used for design space exploration of a cache configuration, where variation in execution time is more important than its absolute value.

Application	Correlation	Application	Correlation
fft	0.96	radix	0.98
raytrace	0.99	cholesky	0.99
water_spatial	0.99	water_nsquared	0.99
barnes	0.99	ocean	0.99
blackscholes	0.99	swaption	0.98
fluidanimate	0.87	x264	0.97
mpegenc	0.99	mpegdec	0.99

TABLE IV
CORRELATION BETWEEN THE NO. OF CACHE MISSES ESTIMATED USING MODIFIED DINERO METHOD AND SIMULATION

Application	Speedup	Application	Speedup
fft	12	radix	35
raytrace	17	cholesky	26
water_spatial	157	water_nsquared	37
barnes	23	ocean	14
blackscholes	29	swaption	62
fluidanimate	218	x264	83
mpegenc	57	mpegdec	27

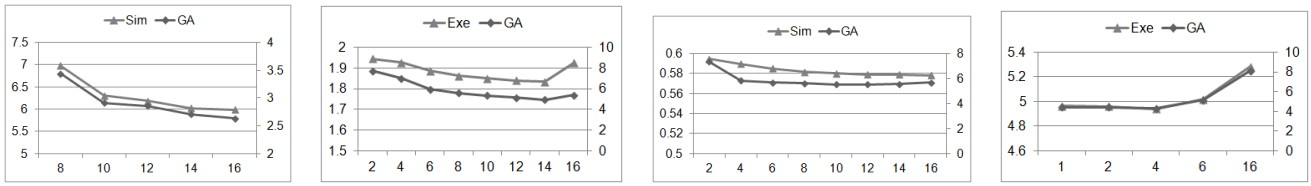
TABLE V
SIMULATION SPEEDUP OBTAINED USING OUR METHOD OVER THE CYCLE-ACCURATE ARCHITECTURAL SIMULATION

C. Accuracy of GA Formulation

Like prior experiments, we modified GA formulation to obtain remap configurations allocating specific number of L2 slices. This method can be considered as an approximation of exhaustive search method. We determined EDP by simulating these configurations. We also determined EDP by simulating the NORC obtained using our GA formulation. EDP of the NORC is minimum for most of the applications and is within 5% of the mimimum EDP obtained for all applications. Fig. 6 gives execution time and EDP obtained with the remap configurations allocating varying number of L2 slices.

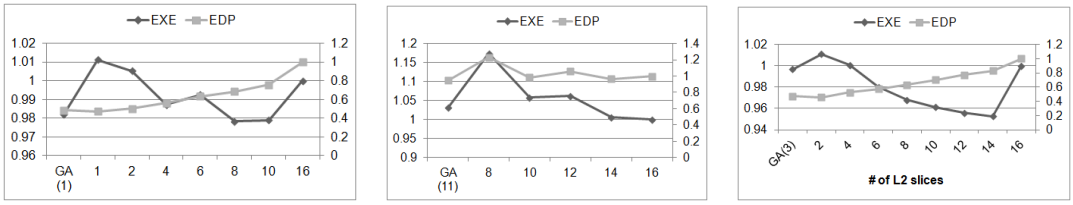
As shown in Fig. 6(a), blackscholes needs only one L2 slice and EDP savings decrease on allocating more than one L2 slices. The remap configuration determined with GA method also allocates only one L2 slice. Similarly, in the case of cholesky (Fig. 6(b)), if only 8 L2 slices are allocated then EDP degrades by 22% and execution time by 17% over the reference SNUCA. It needs at least 10 L2 slices to obtain the maximum EDP savings. However, execution time degrades by 5% on allocating 10 L2 slices. The NORC allocates 11 L2 slices and maps rest of L2 slices to obtain 5% of EDP savings and execution time degrades by 3% only.

It should be noted that our GA approach explores all possible number of powered on L2 slices simultaneously and eventually finds a remap configuration that gives the maximum EDP savings. In dynamic implementation of the remap policy



(a) cholesky (b) mpegenc (c) fluidanimate (d) x264

Fig. 5. The primary Y axis plots execution time (in M) obtained on simulating the remap configurations allocating 2, 4... number of L2 slices. The secondary Y axis plots execution time (in M) estimated using Eq. 7 for the same remap configurations. Execution time estimated using our trace based model shows a good correlation to the trend seen with simulation.



(a) blackscholes (b) cholesky (c) mpegenc

Fig. 6. Graphs show normalized execution time (on the primary Y axis) and EDP (on the secondary Y axis) obtained on simulating remap configurations allocating 2, 4... L2 slices. All values are normalized w.r.t. the reference execution on the SNUCA. "GA(X)" is the near optimal remap configuration obtained with our GA method. This remap configuration allocates "X" number of L2 slices.

[1], we use monitoring interval of 4 million clock cycles to determine significant change in the number of L2 cache accesses made by an application. We call it as the working set size (WSS) of an application. Hence, we also divided a trace based on the significant change in WSS which is evaluated after every 4 million clock cycles for all applications. We determined NORC separately for each trace. EDP savings obtained with this method are lesser than that obtained with the current method which considers the entire trace to determine the NORC. The EDP savings are lower because of overhead incurred due to changes in the remap configuration. Hence, we do not present those results in this paper.

D. Simulation Speedup

Diaz et al. [9] evaluate fitness of a chromosome for a dual application scenario using cycle-accurate architectural simulation, which is very slow. Hence, we evaluate the simulation speedup obtained using our method of estimating fitness over the cycle-accurate simulation of the same remap configuration (Table V). Applications such as water_spatial and x.264 show significant speedup (up-to 218x) as most of the accesses are L1 hits. As a result, a trace between L1 misses and L2 is smaller. Memory intensive applications such as ocean also shows speedup of 14x. Hence, we believe our method can help in solving other cache optimization problems such as cache partitioning and cache design space exploration on CMPs.

VI. Conclusions

Genetic algorithms have been used in various fields such as medical and chip design. Diaz et al. [9] used GA for the first time to solve cache optimization problem in the case of a dual thread scenario. However, the use of architectural simulation to estimate fitness of a chromosome makes it very slow and non-scalable. Hence, we propose a trace based model which shows a simulation speedup between 14x-218x over the cycle-accurate simulation. We also present a detailed validation of our methodology and demonstrate its scalability for multithreaded workloads on a sixteen tiled CMP.

References

[1] A. M. Dani, B. Amrutur, and Y. N. Srikant, "Adaptive Power Optimization of On-chip SNUCA cache on Tiled chip multicore architecture Using Remap Policy," in *WAMCA*, 2011.

[2] C. Zhang, F. Vahid, and R. Lysecky, "A self-tuning cache architecture for embedded systems," *ACM TECS*, 2004.

[3] R. A. Sugumar and S. G. Abraham, "Set-associative cache simulation using generalized binomial trees," *ACM Trans. Comput. Syst.*, 1995.

[4] O. Avissar, R. Barua, and D. Stewart, "An optimal memory allocation scheme for scratch-pad-based embedded systems," *ACM Trans. Embed. Comput. Syst.*, 2002.

[5] M. D. H. Jan Edler, "Dinero trace-driven uniprocessor cache simulator," http://www.cs.wisc.edu/ markhill/DineroIV.

[6] D. K. Tam, M. Stumm, and et al"RapidMRC: Approximating L2 Miss Rate Curves on Commodity Systems for Online Optimizations ," in *ASPLOS*, 2009.

[7] R. L. Mattson, G. J, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," in *IBM Systems Journal*, 1970.

[8] Aamer Jaleel, "Cmpsim: A pin-based on-the-fly multi-core cache simulator," 2008.

[9] J. Díaz, S. López, and et al, "Improving SMT performance: an application of genetic algorithms to configure resizable caches," in *ACM GECCO*, 2009.

[10] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *ASPLOS*, 2002.

[11] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *ISCA*, 1995.

[12] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *PACT*, 2008.

[13] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A tool to model large caches," 2009. [Online]. Available: http://www.hpl.hp.com/techreports/2009/HPL-2009-85.html

[14] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob, "DRAMsim: a memory system simulator," 2005.

[15] R. Nagpal, A. Bhardwaj, and Y. N. Srikant, "Intacte: an interconnect area, delay, and energy estimation tool for microarchitectural explorations," in *CASES*, 2007.

[16] M. lap Li, R. Sasanka, S. V. Adve, Y. kuang Chen, and E. Debes, "The ALPBench benchmark suite for complex multimedia applications," in *IEEE ISWC*, 2005.

[17] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC simulator," 2005, http://sesc.sourceforge.net.

[18] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet general execution-driven multiprocessor simulator (gems) toolset," 2005.