Applying Genetic Algorithms to Optimize Power in Tiled SNUCA Chip Multicore Architectures

Aparna Mandke Bharadwaj Amrutur Y. N. Srikant

IISc-CSA-TR-2010-7
http://archive.csa.iisc.ernet.in/TR/2010/7/

Computer Science and Automation
Indian Institute of Science, India

March 2010

Applying Genetic Algorithms to Optimize Power in Tiled SNUCA Chip Multicore Architectures

Aparna Mandke, Bharadwaj Amrutur, Y. N. Srikant

Abstract

We propose a novel technique for reducing the power consumed by the on-chip cache on SNUCA chip multicore platform. This is achieved by what we call a "remap table", which maps accesses to the cache banks that are as close as possible to the cores, on which the processes are scheduled. With this technique, instead of using all the available cache, we use a portion of the cache and allocate lesser cache to the application. We formulate the problem as an energy-delay(ED) minimization problem and solve it offline using a scalable genetic algorithm approach. Our experiments show up to 40% of savings in the memory sub-system power consumption and 47% savings in energy-delay product (ED).

1 Introduction

The rise of cell phones as the most preferred mobile computing device has necessitated the need for more general purpose solutions on embedded systems - a trend reflected in the introduction of multi-core ARM Cortex Series processors with configurable cache sizes, which lets the designer choose an appropriate cache size based on the use case. The number of cores in embedded processors are increasing due to the increasing number of applications on the mobile platform, making larger caches necessary on these platforms. One of the major challenges in designing efficient cache hierarchy is to have low cache access latency and low cache miss rate, which are often conflicting demands. To strike a balance between them, the huge cache is partitioned into multiple banks and these banks are connected using switched network or crossbar network[8].

Similarly, to make a CMP platform scalable, cores and shared caches are arranged in identical building blocks called tiles (Fig. 1). Tiles are replicated

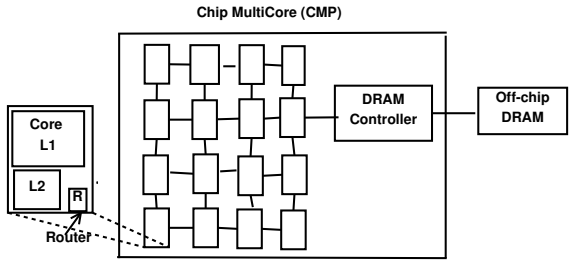


Figure 1: **Fig. shows the tiled architecture used in experiments.**

and connected through an on-chip switched-network (NoC). Each tile has a core, a private L1 instruction and data cache, L2 cache and a router. L2 cache is distributed in all tiles and is shared by all the cores. To maintain cache coherence between the private L1 caches, a directory is present in each tile. These tiles are connected to one another via 2D-mesh switched network and per-tile router.

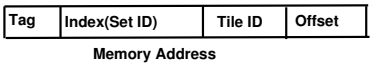
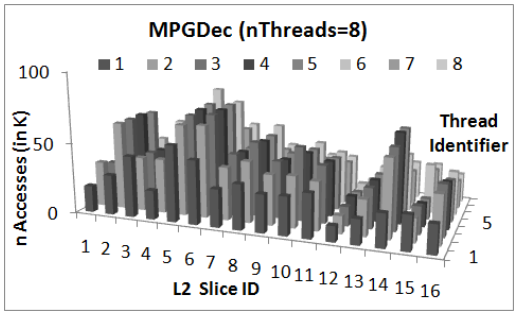


Figure 2: **In SNUCA caches, "Tile ID" bits from the address determine the "home" L2 slice where data is cached.**

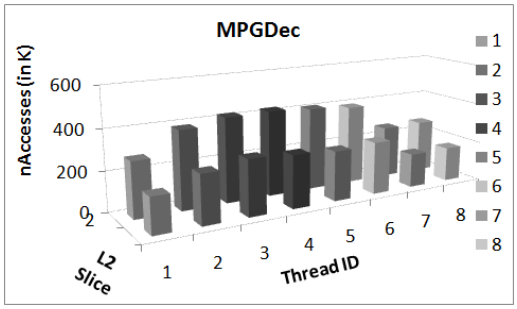
Memory address determines the location of L2 bank where data will be cached, which we call a "home location" of that address (Fig. 2). Data can reside only in its "home" L2 bank. Hence, the architecture is referred as *Static Non-Uniform Cache Access (SNUCA)* architecture.

When an application is executed on such a tiled architecture with distributed but shared L2 cache, accesses get dispersed to L2 slices[1] in all tiles. The use of different L2 slices is not uniform. This can be observed in Fig. 3(a). The graph plots the number of accesses made by 8 threads in MPG Decoder[9] application to different L2 slices present on a sixteen tile CMP platform. The memory accesses done by the cores get dispersed to all L2 slices. Some L2 slices like 11, 12, 15 etc. are underutilized, whose accesses can be merged in nearer L2 slices, thereby saving transit time and leakage power consumed by these L2 slices. We observed not only power savings of 44% but also 2.6% improvement in the execution time when the same application is executed with just two L2 slices, which service accesses made to the rest of the fourteen L2 slices, besides their own. Graph in Fig. 3(b) shows the change in the traffic pattern, when 2^{nd} and 3^{rd} L2 slices service all the L1 misses.

[1]L2 slice means all L2 cache banks in a tile.



(a) **All sixteen L2 slices have been used.**



(b) **Only 2^{nd} and 3^{rd} L2 slices have been used using the remap table.**

Figure 3: **The remap table offers the choice of not only size but also location of the allocated L2 cache.**

We use a novel technique, which we call as a "Remap Table" to merge the accesses made to the under-utilized distant L2 slices to nearer L2 slices. The unused L2 slices can be switched off thereby saving power. In multi-tasking environment, every application will reduce its energy footprint by utilizing lesser L2 cache. Unrelated conflict misses at the shared L2 cache can be avoided by partitioning the cache among all the applications.

For embedded applications like video applications, the remap table can be configured using an offline method by determining parameters which influence the application cache requirement. Cache requirement is independent of input data value if these influential parameters are kept same.

We varied the number of allocated L2 slices in steps of 2, for a sixteen tile CMP platform. For each configuration, we *exhaustively* searched all remap table configurations and evaluated them using simulation. The remap table giving the highest ED savings over the reference, which allocates all the sixteen L2 slices, is desired. Table 1 shows % savings in ED as the number of allocated L2 slices is varied.

On allocating fewer than the required number of L2 slices (2 in this ex-

Table 1: **Table shows % savings in energy-delay when the number of allocated L2 slices is varied with respect to the reference, which allocates all 16 L2 slices. R_2 denotes that two L2 slices are allocated and rest of the fourteen slices are mapped to these two selected slices. %Pf and %ED give % savings in the execution time and energy-delay product of the application, with respect to the reference.**

Config	R_2		R_4		R_6		R_8		R_10		R_12	
	%Pf	%ED	%Pf	%ED	%Pf	%ED	%Pf	%ED	%Pf	%ED	%Pf	%ED
MPGDec-8	-0.52	31	4	32.8	4	28.4	4.1	24	3.36	18	2.67	12.5

ample), the number of offchip DRAM accesses increases. It degrades the execution time of the application when compared to the reference. When more than the required number of L2 slices (4 onwards in this example) are allocated, it causes dispersion of the accesses, increasing time spent in transit. Hence, one needs to allocate the optimum number of L2 slices to the application.

In our problem, not just the number of L2 slices allocated to the application is important but their location is also important. *Clearly, as the number of tiles and hence, the number of L2 slices present on the CMP platform is increased, exhaustive search method will not scale. Hence, we solve this problem using more scalable "Genetic Algorithms".*

Following are our contributions:

- We propose to reduce the footprint of applications with smaller cache requirements than the available cache, using a remap table, which allows the choice of size as well as the position of chosen L2 slices.
- We model the penalty incurred by all L1 misses and the energy consumption of the memory sub-system components, in the presence of a remap table using a trace based approach (Section 3).
- We apply genetic algorithm (GA) to determine the optimum remap table configuration (Section 3.5). We observe that our remap table strategy gives power and energy-delay savings up to 40% and 47% without compromising much of the performance. Such an offline determination of the remap table is applicable for embedded platforms.
- The simulation framework models power consumption of all memory sub-system components like offchip DRAM, interconnect and cache, unlike previous research. (Section 4).

Our experimental results, related work and conclusions are described in sections 5, 6 and 7, respectively.

2 Architectural Changes

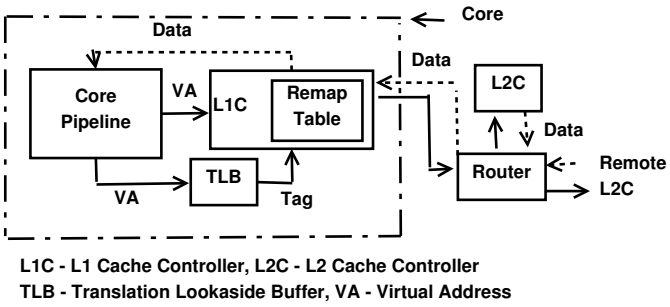


Figure 4: **Block diagram explaining working of the remap table strategy. The remap table is configured in the L1 Cache Controller.**

We propose to remap certain subset of L2 slices onto the remaining ones. *When one L2 slice is mapped to another slice, everything else remains the same including the core on which threads are executing and hence location of their L1 caches. Only the accesses made to the L2 slice in one tile are remapped to L2 slice in another tile.*

The remap table can be implemented as part of an L1 controller and is basically a hardware register file. On an L1 miss, L1 controller will set a destination L2 slice to the remapped L2 slice, before sending a memory request over the network, as shown in Fig. 4. The remap strategy is independent of whether L1 is physically tagged or virtually tagged. This is also independent of the position of tile id bits in the memory address. This is because, virtual to physical address translation (TLB Lookup) happens before sending the request to the L2 over the network. Hence, the destination L2 slice bits are always available during the remap table lookup. If bits next to the cache line offset are used as L2 slice id bits,as shown in Fig. 2, then the L1 controller can perform remap lookup at the same time, when L1 cache is checked for a hit. In case of an L1 miss, L1 controller can always lookup the remap table simultaneously while creating the outgoing memory request. This ensures that the remap strategy does not add any additional latency on L1 miss.

2.1 Obtaining Remap Table

Typically on embedded platforms the applications are known apriori and are well characterized. Hence, we derive the remap table based on offline profiling of these applications. This approach is well established for these class of systems. The working set and hence memory requirement for applications like FFT does not change if the number of points on which FFT operation is performed, remains the same. Memory requirement is independent of the actual data. The application can be parameterized and the remap table can be specified for each set of input parameters. For example, a different remap table can be used for different FFT window sizes. But for the same window size, the remap table works well for any data input. Applications can provide a callback function which will be invoked by the process scheduler to select the right remap table depending upon the input parameter values. The remap table is part of the process context and can be accessed after process/thread switch. When a new application gets scheduled, its predetermined remap table will be loaded.

3 Problem Formulation

It can be observed from the Table 1 that, on allocating fewer than the required number of L2 slices, leads to power savings at the cost of execution time of the application. Allocation of more than the required number of L2 slices, causes dispersion of L2 accesses. This increases the time spent in transit, leading to increased execution time of the application. This also increases power consumption due to increase in cache leakage power. Thus, there exists an optimum number of L2 slices for which application gives the least energy-delay product. Eq. 1 gives the objective function which we want to minimize.

$$EDP = (E_{cache} + E_{interconnect} + E_{DRAM}) * Time \tag{1}$$

The meaning of the various terms is explained in Table 2. In this study, we have considered multi-threaded workloads. We assume that there is a one-to-one mapping between threads and cores. This assumption is in line with other work done for CMP platforms.

3.1 Estimation of Energy Consumption

- *E_cache* : Leakage power is directly proportional to the number of allocated L2 slices. Leakage power per L2 slice is obtained using CACTI

Table 2: **Table gives the meaning of the various terms, used in genetic algorithm based problem formulation. The last column describes how these terms are obtained.**

Parameter	Description	Method of Determination
EDP	Energy-Delay Product	Eq. 1
E_cache	Energy consumption of the cache	CACTI[13]
$E\_interconnect$	Energy consumption of interconnect	-
E_DRAM	Energy consumption of off-chip DRAM	Statistical model of data obtained using DRAMSim[15]
$Time$	Time taken by the application to complete its execution	Eq. 3
C_{total}	Total time spent in servicing all L1 misses	Eq. 8
C_{remap}	Total time spent in servicing all L1 misses for a given remap table	Eq. 9
K	# of threads used by the application for execution	Specification
N	# of tiles/L2 slices on the CMP platform	Specification
S	# of L2 slices allocated to the application	GA (Section 3.5)
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[12]
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 3.5)
p_j	Penalty to service a single cache miss in L2 slice in tile j. This penalty includes the latency of the memory access and the network latency between L2 cache in tile j and the memory controller.	Intacte, an average of variable 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 3.5)
K_c	Constant decides priority given to the cache miss penalty	Empirically
K_t	Constant decides priority given to the transit time penalty	Empirically

[13]. The total dynamic energy dissipated in L2 slices remains nearly the same for all remap table configurations. Hence, we ignore dynamic energy consumption in the objective function.

- $E\_interconnect$: Interconnect power does not vary by large amount if the application performance is comparable to the reference. Hence, we ignore this factor as well in the objective function.

- E_DRAM : Off-chip DRAM power consumption depends upon the number of memory accesses, row management policy and address correlation between the sequence of accesses. We performed simulation on various applications by varying L2 slice allocation and obtained off-chip DRAM power consumption. We empirically observed that the memory power consumption remains almost same if the number of DRAM accesses are comparable to the reference. This is because DRAM operates at much lower frequency (667 MHz) than the cache/core frequency (3 GHz). As we use closed row management policy, we can ignore correlation between the consecutive addresses in the memory request stream. *We use an approximate conservative memory power consumption model based on linear interpolation of the experimental data.* Such a model, can be obtained for every application based on its DRAM power consumption.

  If some remap configuration estimates considerably more number of memory accesses than the reference then the memory model will discard that solution. If the number of memory accesses are low due to over-allocation of L2 slices then the leakage power of the cache will discard that solution. This ensures that our model allocates right amount of the L2 slices. Hence, the approximate DRAM power consumption model suffices in our case. EDP can be simplified as in Eq. 2.

$$EDP \approx ((LeakagePwrPerL2Slice * \#OfL2SlicesAllocated + memPwr) * Time) * Time \quad (2)$$

3.2 Estimation of Time

To calculate $Time$ component of Eq. 2, we need to estimate the time taken by the application to complete its execution. Time is expressed in terms of the processor clock cycles. Execution time is proportional to the time spent in servicing L1 cache misses. Lesser the number of cache misses, lesser will be execution time of the application, and hence the EDP product. Thus $Time$ in Eq. (2) is,

$$Time \propto C_{remap} * 1/ProcessorFrequency \tag{3}$$

3.3 Estimation of Transit Time Penalty

Total time spent by 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,

$$\tau_i = \sum_{0 \leq j < N} \eta_{ij} L_{ij} \tag{4}$$

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

$$\tau_= \sum_{0 \leq i < K} \tau_i = \sum_{0 \leq i < K} \sum_{0 \leq j < N} \eta_{ij} L_{ij} = \sum_{0 \leq j < N} \sum_{0 \leq i < K} \eta_{ij} L_{ij} \tag{5}$$

All the threads execute concurrently. Hence transit time spent by these threads overlaps with one another. Summation of their timings just denotes maximum time the application might spent in transit. We have also ignored transit time latency due to the coherence traffic. We consider data parallel applications, hence coherence traffic is negligible compared to the other traffic.

3.4 Estimation of Cache Miss Penalty

Execution time of the application is greatly affected by the cache miss rate, especially in the lowest level cache(LLC) in the cache hierarchy. So the remap table entries should be chosen such that the total number of L2 cache misses should not increase tremendously. Cache miss cost c_j of L2 slice j is,

$$c_j = \sum_{0 \leq i < K} p_j . \alpha_{ij} \tag{6}$$

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

$$C_{Miss} = \sum_{0 \leq j < N} c_j = \sum_{0 \leq j < N} \sum_{0 \leq i < K} p_i . \alpha_{ij} \tag{7}$$

Hence, the total cost incurred by misses in all L1 caches is the sum of costs given by Equation (5) and Equation (7).

$$C_{total} = \sum_{0 \leq j < N} \sum_{0 \leq i < K} \eta_{ij} L_{ij} + \sum_{0 \leq j < N} \sum_{0 \leq i < K} p_j . \alpha_{ij} \tag{8}$$

We choose the remap table configuration such that C_{total} is minimized, as the application execution time is proportional to the time spent in L1

misses. Of course, it also depends on how many compute operations can be overlapped with these L1 misses.

We give more weight to the cache miss penalty since the latency of one miss in LLC is much higher than its transit time penalty. We empirically found that $K_t = 1$ and $K_c = 8$, remap unallocated L2 slices more uniformly to the allocated L2 slices. We use these values of K_c and K_t in all our experiments. Hence, for a given remap table, the cost will be,

$$C_{remap} = K_t \sum_{0 \le j < N} \sum_{0 \le i < K} \eta_{ij} L_{i\delta(j)} + K_c \sum_{0 \le j < S} \sum_{0 \le i < K} \alpha_{ij} p_j \tag{9}$$

3.5 Genetic Algorithm Formulation(GA)

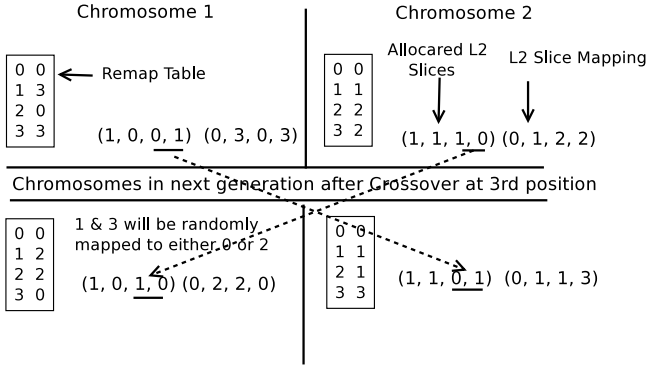


Figure 5: **The remap table configuration is encoded as a chromosome. Fig. shows the crossover operation.**

Genetic algorithms are a class of randomized search algorithms useful in function optimization. A possible solution of the problem is encoded as a binary fixed size array, called chromosome. We use a chromosome to represent one remap table configuration. As shown in Fig. 5, the remap table is represented as 2 strings. 1^{st} binary string gives the L2 slices allocated to the application and 2^{nd} gives the mapping of unallocated slices. E.g. a chromosome "(1,0,0,1)(0,3,0,3)" denotes that 0^{th} and 3^{rd} L2 slices are allocated. 1^{st} and 2^{nd} L2 slices are mapped to the 3^{rd} and 0^{th} slices, respectively. Following are the details of application of the genetic algorithm approach:

- The fitness value of the chromosome is given by EDP in Eq. 2.
- The population size of 100 chromosomes

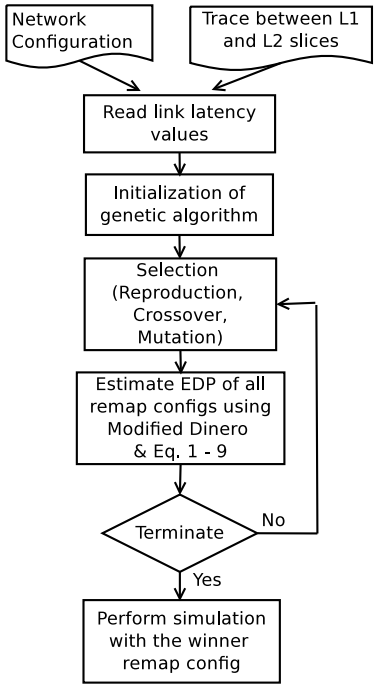


Figure 6: **The flow chart explaining our problem formulation using genetic algorithm approach.**

- randomly selected single point crossover between two chromosomes with probability of 0.65. It is done only on first binary string which represents the allocated L2 slices. Unallocated L2 slices are randomly mapped to the allocated L2 slices. Each remapped slice is mutated to another allocated L2 slice with mutation probability of 0.015. We have chosen mutation and crossover probability values within the standard range.

- Generally, in the case of genetic algorithms, higher the fitness value of the chromosome, better is the solution. But in our problem, lower the EDP value, better is the remap configuration. To convert our minimization problem into a maximization problem, we find out the largest fitness value of the population and deduct fitness values of all the other chromosomes from it. We use Roulette Wheel selection method.

- We terminate our search algorithm after 50 generations and use the fittest solution for further evaluation. *For all applications, GA stabilizes after 20-30 generations.*

The flow chart in Fig. 6 gives the steps to be followed to determine the near optimal remap table configuration. Now we describe, how we calculate all the terms, required to obtain the fitness value given by Eq. 2. To determine the number of cache misses in L2 cache for a given remap table, we use a modified trace-driven Dinero cache simulator[6]. We simulate an application with all L2 slices and then capture the traffic generated between all L1 and L2 caches in a trace. The modified Dinero simulator takes this trace and an instance of a remap table as input. It gives the number of cache misses per L2 slice. On CMPS, shared caches are noninclusive. So conflict misses in L2 do not cause evictions in L1 cache. Most of the accesses are L1 hits. Due to these reasons, this approximate method gives quite reasonable estimation of L2 cache misses. The same trace is used to determine the transit time cost.

3.6 Accuracy of Time Estimation Function

To check the accuracy of our cost modelling method, we modified GA algorithm to obtain the remap table configuration which allocates 2, 4, 6, 8, 10, and 12 L2 slices. The remap table configuration obtained using this method, is simulated. C_{remap} value is obtained using simulation and by the cost modelling method described in previous subsection. The statistical correlation between these two costs is above 80% for all the applications.

4 Experimental Evaluation

4.1 Applications used in experiments

We evaluate multi-threaded workloads with one-to-one mapping between the threads and cores (Table 3). This assumption is in line with other work done for CMP platforms. The technique described in this paper is also applicable to multitasking workloads, where all single threaded applications run on different cores. We have taken readings for various number of threads since application may not create sixteen threads due to limited data or task parallelism.

4.2 Experimental Setup And Methodology

We use a framework, called as "Sapphire" which models all the system components accurately[1]. Sapphire uses SESC[14] to simulate a core, Ruby component from GEMS [11] to simulate the cache hierarchy and interconnects. DRAMSim[15] is used to model the offchip DRAM and also estimates

its power consumption. Intacte [12] is used to estimate the low level parameters of the interconnect such as the number of repeaters, wire width, wire length, degree of pipelining and also the power consumed by the interconnect. Power consumed by the cache components is estimated using CACTI [13].

In order to estimate the latency (in cycles) of a certain wire, we need to create the floorplan of the tile and estimate the wire length. The floorplan determination requires estimating area of all components in the tile:

- core : This is estimated based on the area of a core of Intel Nehalem (since we could not obtain an estimate of a MIPS core). The core area of Intel Nehalem can be considered as a pessimistic estimate of the MIPS core.

- cache : The area occupied by the L2 cache is obtained using an approximate estimate of 7.5mm^2 per MB of cache, which is the density achieved on the AMD Shanghai processor. The L1 cache is of size 32KB whose area is very small and is included in the processor area and hence is not depicted in the Fig. 7.

- router : The area of the router is quite negligible at 32nm.

The floorplan of a typical tile is shown in Fig 7. The wire lengths are determined for the floorplans at a frequency of 3GHz and 32nm technology.

Table 3: **Applications used for experimentation**

Application	# of Threads	Description
Alpbench Benchmark		
MPGEnc	8,16	Encodes 15 Frames of size 640x336
MPGDec	8,16	Decodes 15 Frames of size 640x336
Parsec Benchmark		
H.264 Encoder	8	Uses pipeline parallelism, 1B instructions with ref i/p
Splash2 Benchmark		
FFT	8,16	FFT on 64K points
LU (Non-Continuous)	8,16	256x256 Matrix Factorization,B=16
Cholesky	8	Cholesky Factorization

Table 4: **System configuration used in experiments**

Core	out-of-order execution, 3GHz frequency, issue/fetch/retire width of 4
L1 Cache	32KB, 2 way, 64 bytes cache line size, access latency of 2 cycles (estimated using CACTI[13]), private, cache coherence using MOESI
L2 Cache	1MB/tile, 8 way, 64 bytes cache line size, access time of 4 cycles (estimated using CACTI), shared and distributed across all tiles
Interconnect	16 bits flit size, 4x4 2D MESH, deterministic routing, 4 virtual channels per port, credit based flow control
Off chip DRAM	4GB in size, DDR2, 667MHz freq, 2 channels, 8 bytes channel width, 8 banks, 16K rows, 1K columns, close page row management policy

These are given in the table shown in Fig. 7. The latency of a link in clock cycles is equal to the number of its pipeline stages. Some of these latencies are quite high and hence cannot be ignored during architectural simulations. To obtain the power consumption of interconnect, we compute the link activity and coupling factors of all links, caused due to the request and response messages sent over the network.

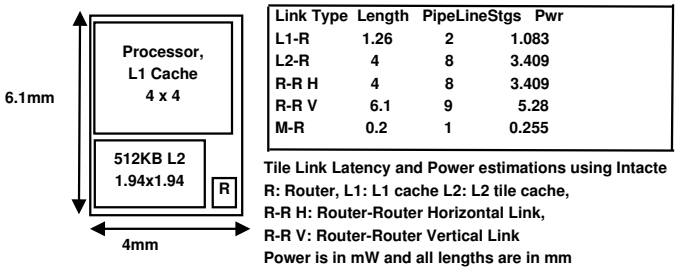


Figure 7: **The floorplan of a tile with L2 slice of 512KB in size.**

Table 4 gives the system configuration used in our experiments.

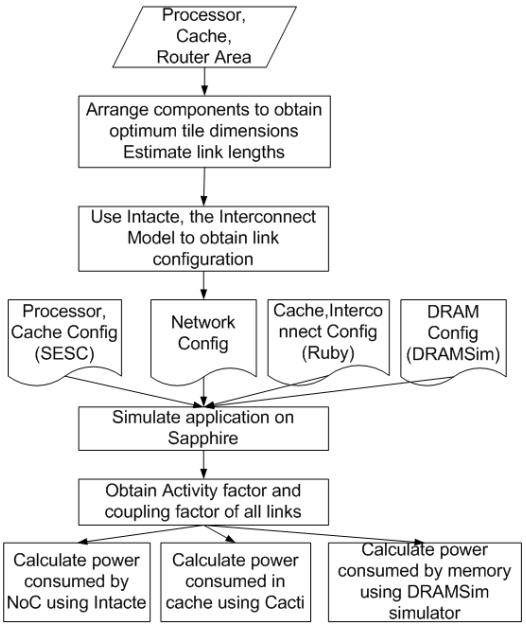


Figure 8: **The steps to be followed to perform the detailed power-performance analysis on Sapphire.**

4.3 Simulation Procedure

The flow chart in Fig. 8 shows the experimental procedure. It includes computing the area of tile components, computing link lengths and low level link parameters using Intacte and then performing simulation on Sapphire. Sapphire estimates the activity and coupling factors of all the links. Intacte determines the power dissipated in the interconnects using these activity factors. The power consumed by the off-chip DRAM and the on-chip cache is estimated using DRAMSim and CACTI power model, respectively.

5 Results

Some decisions like code generation or allocation of variables to scratch pad are usually taken based on the profiled information at compile time for embedded platforms. We have also solved this problem using a similar approach. The remap table can be determined using few frequently occurring representative inputs. Most conservative remap table, i.e. one which allocates more number of L2 slices should be used. The remap table could be specified for a specific input range. If the input parameters are out of all the calibrated ranges then the default remap table which maps every L2 slice to itself (i.e.

Table 5: **Table shows the % savings in the execution time, power and energy-delay product of the application over the reference. FFT-8 indicates FFT application is executed with 8 threads.**

App-nThrds	I/P	# of L2 slices	%Pf	%Pw	%ED
64K points FFT - **Value of points is varied**					
FFT-8	I1*	9	3.55	20	27.26
	I2		3.56	20.1	25
	I3		3.6	20	25
FFT-16	I1*	10	6.6	15.58	28.13
	I2		2.31	16.4	19.6
	I3		2.4	16.5	19.8
256x256 Matrix Factorization - **Value of points is varied**					
LUCN-8	I1*	5	6.4	22.5	32.9
	I2		1.7	23.4	24.9
LUCN-16	I1*	5	3	16.9	22.5
	I2		-0.8	18.5	16
128x128 Matrix - **Change in Matrix Dimensions**					
LUCN-8	I1	5	-0.5	25.5	23.6
	I2		0.09	25.3	24.5
LUCN-16	I1	5	-0.9	19.6	17.1
	I2		-1.1	19.7	16.8
Cholesky-8	d570*	9	3.39	7.454	13
	tk23		2.46	1	5.79
	tk29		-0.2	7.8	6.8
	lshp		3.7	8.82	14.9

Table 6: **Table shows the % savings in the execution time, power and energy-delay product of the application over the reference.**

Config	Input	# of L2 slices	% Pf	%Pw	% ED
Change in frame resolution and input image					
Dec-8	I1* - 15 Frames of res. 640x336	4	4.2	26.6	33
	I2 - 3 Frames of res. 128x218		4.6	31	36
	I3 - 20 Frames of res. 704x480		2.7	21.8	24.7
	I4 - 20 Frames of res. 706x576		2.9	21	24.3
Dec-16	I1* - 15 Frames of res. 640x336	3	3	22.6	27.9
	I2 - 3 Frames of res. 128x218		6.2	34	38
	I3 - 20 Frames of res. 704x480		-1.2	16	12
	I4 - 20 Frames of res. 706x576		-0.6	14.6	12.15
Enc-8	I1* - 15 Frames of res. 640x336	7	3.2	19.3	25.6
	I2 - 3 Frames of res. 128x218		4.9	23.6	30
	I3 - 10 Frames of res. 704x480		-1.6	16	12
Enc-16	I1* - 15 Frames of res. 640x336	10	6.8	8.7	22
	I2 - 3 Frames of res. 128x218		5.6	16	21
	I3 - 10 Frames of res. 704x480		1	7.2	8,4
X264-6	I1* - Frames res. 640x360, 1B. inst.	3	7	40.6	49.3
	I2 - Frame res. 384x288, 2B insts		4.16	37.9	41.6
	I1 - GOP changed from IBBBP to IBBBBBP		6.95	40.27	46.99
	I3 - Frame res. 352x240 2B insts.		0.8	39.6	39

remapping disabled) should be used.

For applications like FFT, memory requirement depends on the number of points on which FFT is performed and not on the actual values of these points (Table 5)[2]. We varied values of these points and performed simulation using the same remap table, obtained with the reference input. % savings in execution time, power and energy-delay do not change even on changing the input data set.

In case of LU, which performs matrix factorization, the remap table can be specified for the range of matrix dimensions. Power and energy-delay savings obtained are primarily dependent on dimensions of the matrix. They are not influenced greatly by the actual values of the matrix elements. Similarly, Cholesky factorization is performed on sparse matrices which are symmetric and positive-definite. For input of tk29, this benchmark shows degraded execution time primarily, on account of 39% increase in memory accesses. However, this is compensated by 16% reduction in the transit time, leading to a marginal performance loss of 0.2% while achieving power savings of 7.8% over its reference. Even though, marginal degradation in execution time is observed, power savings are more important on the embedded systems.

For video applications like MPEG decoder, encoder and H.264, one can vary most influential parameters like frame resolution, bit rate etc and configure remap table for various frame resolutions. For such applications, working set does not change drastically even on changing the number of frames. For example, we determined the remap table using input image with frame resolution of 704x480 for MPG Decoder and Encoder application and took readings for images with different frame resolutions. The same remap table continues to give power savings as shown in Table 5. In case of H.264 Encoder, we varied the number of B frames between I and P from 3 (IBBBP) to 5 (IBBBBBP). H.264 encoder uses pipeline parallelism. The mentioned change increases the number of concurrent threads of execution; even then the same remap table achieves improvement in execution time of 6.95% with power and energy-savings of 40% and 47% respectively, over its reference. These experiments indicate that one can certainly determine the important performance determining parameters of media applications and configure the remap table using an offline method.

6 Related Work

Lot of research has been done in the past, to reduce the L2 bank access latency in the context of large NUCA caches [7]. In [8] and [3], authors

[2]In table 5 and 5, * indicates that the remap table is evaluated using this input.

propose to migrate the cache lines near to the core within the same set whose ways are spanned across different banks (DNUCA architecture - Dynamic Non-Uniform Cache Access). None of these explicitly optimize the power consumption of caches. In [2], Bardine et. al optimize dynamic power consumption of DNUCA cache by avoiding unnecessary cache line promotion/demotion. However, they mainly study a single core scenario. In [10], Javier et. al propose a replacement policy for DNUCA architecture which gives performance improvement of 8% and energy savings of 4%. In [5], authors conclude that the DNUCA cache is not justified for CMPs due their energy consuming bank accesses, complex lookup and replacement logic, also cache lines may ping-pong in different directions due to concurrently executing threads. Hence, we target power optimization in SNUCA caches.

In [4], Diaz Josefa and et. al have used genetic algorithms for the first time, for the cache resizing problem. They evaluate the fitness of the solution using architectural simulation, which is very slow. Hence, they can use only 30 chromosomes in the population. We overcome this drawback by modelling the fitness function by a faster method.

7 Conclusions and Future Work

In this paper, we propose and implement a technique for reducing the power consumption of an on-chip cache on a SNUCA chip multicore platform. We use a remap table to achieve this. To arrive at appropriate table sizes and entries, we formulate the problem as an energy-delay minimization problem and solve the problem offline using a scalable genetic algorithm. Our technique results in impressive power savings up to 40% and energy-delay savings up to 47%. While the current technique targets embedded systems, it can be extended to configure the remap table dynamically for use in multicore desktop systems as well. We propose this as a direction for future work. Results obtained using the genetic algorithm approach, will help us to analyze the effectiveness of the dynamic approach as well.

References

[1] M. Aparna, V. Keshavan, T. Basavaraj, A. Bharadwaj, and Y. N. Srikant. Sapphire: A framework to explore power/performance implications of tiled architecture on chip multicore platform. Technical Report IISc-CSA-TR-2010-07, Computer Science and Au-

tomation, Indian Institute of Science, India, 2010. URL: http://aditya.csa.iisc.ernet.in/TR/2010/07/.

[2] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, and C. A. Prete. A power-efficient migration mechanism for D-NUCA caches. In *DATE*. IEEE, 2009.

[3] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache architectures. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003.

[4] J. Díaz, J. I. Hidalgo, F. Fernández, O. Garnica, and S. López. Improving SMT performance: an application of genetic algorithms to configure resizable caches. In *ACM Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*, 2009.

[5] J. Huh, C. Kim, and S. W. Keckler. A NUCA substrate for flexible cmp cache sharing. In *ACM Proceedings of the 19th annual international conference on Supercomputing*, 2005.

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

[7] M. Javier, P. Valentin, and G. Jose. ESP-NUCA: A low-cost adaptive non-uniform cache architecture. In *HPCA '10: IEEE Proceedings of the 16th International Symposium on High Performance Computer Architecture*, 2010.

[8] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *ACM SIGPLAN NOTICES*, 2002.

[9] M.-L. Li, R. Sasanka, S. A.-K. Chen, and E. Debes. The Alpbench benchmark suite for complex multimedia applications. In *IEEE International Symposium on Workload Characterization*, 2005.

[10] J. Lira, C. Molina, and A. González. The auction: optimizing banks usage in non-uniform cache architectures. In *ACM Proceedings of International Conference on Supercomputing*, 2010.

[11] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. 2005.

[12] R. Nagpal, A. Madan, A. Bhardwaj, and Y. N. Srikant. Intacte: an interconnect area, delay, and energy estimation tool for microarchitectural explorations. In *ACM Proceedings of the 2007 international conference on Compilers, architecture, and synthesis for embedded systems*.

[13] N. P. Reinman, Glen; Jouppi. CACTI 5.3: An integrated cache timing and power model, compaq, wrl, research report 2000/7. 2000.

[14] 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.

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