Adaptive Power Optimization of On-chip SNUCA Cache on Tiled Chip Multicore Architecture using Remap Policy

Aparna Mandke Bharadwaj Amrutur Y. N. Srikant

IISc-CSA-TR-2011-2
http://archive.csa.iisc.ernet.in/TR/2011/2/

Computer Science and Automation
Indian Institute of Science, India

May 2011

Adaptive Power Optimization of On-chip SNUCA Cache on Tiled Chip Multicore Architecture using Remap Policy

Aparna Mandke, Bharadwaj Amrutur, Y. N. Srikant

Abstract

Advances in technology have increased the number of cores and size of caches present on chip multicore platforms(CMPs). As a result, leakage power consumption of on-chip caches has already become a major power consuming component of the memory subsystem. Hence, we propose to reduce leakage power consumption in static nonuniform access cache(SNUCA) on a tiled CMP by dynamically varying the number of used cache slices and shutting off unused cache slices. A cache slice in a tile includes all cache banks present in that tile. Switched-off cache slices are remapped considering the communication costs to reduce cache usage with minimal impact on execution time. This saves leakage power consumption in switched-off L2 cache slices. On an average, the remap policy achieves 41% and 49% higher EDP savings than static and dynamic NUCA(DNUCA) cache policies on a scalable tiled CMP, respectively.

1 Introduction

Due to advances in technology, the number of cores and on-chip cache size on CMPs has increased. As a result, leakage power dissipated by large caches has become a major power consuming component of the memory subsystem. For power and performance reasons, a large cache is partitioned into multiple banks which are connected using a switched-network[1]. The predetermined bits from a memory address determine the location of L2 bank, where data is cached. We refer that L2 bank as its "home location". Such cache offers non-uniform access latency to various cores on CMP. Hence, the architecture is referred as *Static Non-Uniform Cache Access (SNUCA)* architecture.

The tiled architecture that we consider is shown in Fig. 1. In this architecture, tiles are replicated and connected through an on-chip switched-network(NoC). Each tile has a core, a private L1 instruction and data cache, a slice of L2 cache and a router. L2 cache is distributed across all tiles and shared by all the cores. To maintain cache coherence between private L1 caches, a directory information is present in each tile. These tiles are connected to one another via 2D-mesh NoC and per-tile router.

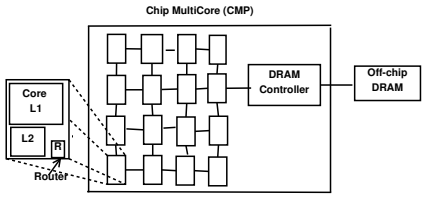


Figure 1: **Tiled SNUCA CMP used for experimentation.**

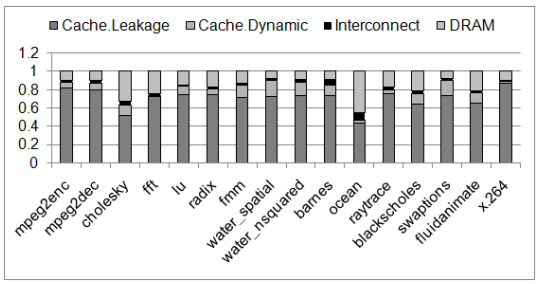


Figure 2: **Graph shows power consumed by various memory subsystem components**

We executed applications from Splash2[2] and Parsec[3] benchmarks suite which spawn 16 threads on a 16 tiled CMP platform. Fig. 2 shows power consumed in various components of the memory subsystem, normalized with respect to(wrt) total memory subsystem power consumption. Following conclusions can be drawn from Fig. 2:

1. Leakage power dissipated in on-chip cache is a major power component for most of the applications.

2. In ocean, power dissipated in DRAM is almost equal to that in cache. It means that its working set size(WSS) is greater than the cache size.

3. For other applications, power dissipated in DRAM is negligible. It means that WSS of these applications is lesser than the cache size.

Hence, by allocating just the required amount of cache and switching off the extra cache, leakage power dissipated in cache can be saved without affecting performance adversely. In our experiments, we additionally found that, accesses made by different threads are dispersed over all L2 slices, which offer non-uniform cache access latencies to the cores. Hence, Kim et al.[1] proposed DNUCA cache to reduce access latency. In DNUCA, the whole address space is mapped onto a column. The predetermined bits from a memory address, determine the row in which data is cached. All L2 slices in a row form a bankset and data can be cached in any of these L2 slices. On L1 miss, data is first searched in the nearest L2 slice and then in rest of the L2 slices in that row before reading it into the nearest L2 slice from memory. Data is migrated towards a nearer slice depending on the number of accesses made by the core. In case of multi-threaded workloads with threads sharing a lot of data, shared data might migrate in conflicting directions, degrading execution time of an application. To avoid unnecessary migrations, we implement 2-bit saturation counter for every cache line. SNUCA is preferred over DNUCA due to its simple lookup and replacement logic. Hence, we propose to reduce leakage power in SNUCA using a "remap policy". Our main contributions are:

1. A novel heuristic to estimate cache requirement on CMPs and reduce leakage power consumption in on-chip cache by selectively switching off L2 slices at run-time using the remap policy, and its implementation.

2. Significant advancements to the state of art of the SESC[4] simulator by enhancing it with 2D mesh NoC and detailed off-chip DRAM models.

We compare energy-delay product(EDP)[1] savings obtained with the remap against SNUCA, DNUCA and also against the drowsy cache scheme[5] applied to SNUCA and DNUCA. Experimental evaluation shows that the remap achieves on an average 41% and 49% higher EDP gains compared to the SNUCA and DNUCA cache policies, respectively.

We review related work in section 2, describe the remap policy in section 3. Section 4 describes experimental setup. Results and conclusions are presented in section 5 and section 6, respectively.

2 Background and Related Work

Here, we mention only those studies which specifically attempt to reduce power consumption in caches. Decay cache [6] estimates *dead time* of a cache

[1]EDP shows a trade-off between energy consumption and execution time.

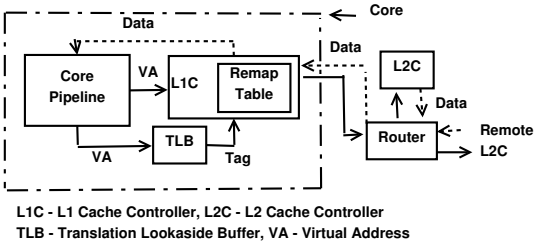


Figure 3: **Block diagram explaining working of the Remap policy.**

line and switches it off after its last estimated use. It operates at granularity of a cache line and has very high hardware overhead. Flautner et al. [5] proposed *drowsy cache* in which less frequently used cache lines are put into drowsy mode. In drowsy mode, supply voltage of a cache line is reduced to dissipate lesser leakage power without losing data. When a cache line is accessed in drowsy mode, additional penalty of 1 cycle is incurred. The drowsy cache technique achieves large energy gains with negligible performance degradation. We have implemented drowsy cache technique in SNUCA and DNUCA cache at the subbank level and monitor drowsy state after every 4000 cycles. Gammie et al. [7] use the ratio of leakage power dissipated in the drowsy state to that in the normal state as 4 at 45nm technology. We also conservatively use the same ratio at 32nm technology. We refer these implementations as "SNUCA.Drowsy" and "DNUCA.Drowsy" in rest of the paper. However, on decreasing supply voltage of a cache line, reduces its soft-error fault tolerance [8]. The remap policy is orthogonal to both drowsy and decay caches. D. Albonesi [9] proposed to disable over-allocated ways of a set-associative cache to reduce its *dynamic* power consumption. Bardine et. al [10] use the same technique to reduce static power consumption of DNUCA cache. They primarily study DNUCA with one or dual cores processor. Their heuristic cannot be applied to a scalable tiled CMP platform. Hence, we do not compare it quantitatively against remap in the current work. In [11], we configure the remap table using offline genetic algorithms. Here, we determine it at run-time according to WSS changes in an application.

3 Remap Policy Details

3.1 Architectural Changes

We propose to remap a certain subset of L2 slices onto the remaining ones. *When one L2 slice is mapped to another slice, everything else remains*

Algorithm 1 Evaluates the remap table

1: Get the number of activeBits(A) and activeLineReplaced(R) of all enabled L2 slices.
2: $EWSS = A + R$
3: $numL2Slices = EWSS/sizeOfL2Slice$
4: $replacementsPerSet = R/totalSets$
5: **if** $replacementsPerSet > 0$ **then**
6: $\quad numL2Slices = numL2Slices + 1$
7: **end if**
8: allocateSlices($numL2Slices, averageLatencyList$)
9: sortUnallocatedSlices()
10: mapUnallocatedSlices()

same including core on which threads are executing and hence location of their L1 caches. Only accesses made to the L2 slice in one tile are remapped to the 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, the L1 controller sets a destination L2 slice to its remapped L2 slice, before sending request over NoC, as shown in Fig. 3. The remap policy is independent of whether L1 is physically or virtually tagged. This is also independent of position of tile id bits in memory address[2]. This is because, virtual to physical address translation (TLB lookup) happens before sending the request to L2 over NoC. Hence, the destination L2 slice bits are always available during the remap table lookup. The remap table access latency can easily be hidden, while creating out-going request to the destination L2 slice.

3.2 Estimation of cache requirement

One active bit per cache line and one 32-bit saturation counter per L2 cache slice is maintained to estimate cache requirement. When a cache line is accessed, its active bit is set. When a cache line with active bit set is replaced, the replacement counter present in that L2 slice is incremented. At the end of a monitoring period, the number of set active bits is counted. The sum of the total number of set active bits(A) and value of replacement counter(R) in all L2 slices gives us cache working set size(WSS).

For cache of 512KB in size and 64B cache line, 8K active bits are needed which is just 0.1% overhead. Our experimental analysis shows that a 32 bit saturating counter is sufficient for our monitoring period, as active bits

[2]We assume that the position of tile identifier bits is between tag and set index bits.

and active lines replacement counters are reset in the beginning of every monitoring period. *Our method adds minimal overhead in terms of hardware area and power consumption.*

Too small monitoring period would set fewer active bits. This would be misinterpreted as lesser cache WSS. On contrary, too long monitoring period would estimate higher WSS, failing to switch-off excess L2 slices. Hence, determination of accurate monitoring period is very important. To estimate it, we measured time in terms of clock cycles between two consecutive accesses made to the same address by any thread in an application. We call it as *reuse time*. Table 1 shows the percentage of accesses with reuse time less than 2M cycles.

MPGEnc	97.7	MPGDec	99
Cholesky	97.5	FFT	94.5
LU	99	Radix	97
FMM	96	Water-Spatial	99
Water-Nsquared	98.9	Barnes	99.5
Ocean	98	Raytrace	98.3
Blackscholes	100	Swaptions	99
Fluidanimate	95	H.264	98

Table 1: Percentage of accesses with reuse time less than 2 million clock cycles.

For all applications that we consider, on an average, 96% of accesses have reuse time less than 2M cycles. Hence, if a cache line is not accessed in the last 2M clock cycles then most likely, it will not be accessed in next 2M clock cycles as well and, it can be evicted. Hence, we use monitoring period of 2M clock cycles.

3.3 Adaptive Implementation of the Remap Policy

L2 slices are allocated adaptively, according to an application's WSS and unallocated L2 slices are mapped onto the allocated ones so that time spent in transit reduces, using the remap table. The new remap table configuration is evaluated by executing Algorithm 1 after every 2M clock cycles. The number of L2 slices to be allocated is equal to the estimated WSS(EWSS) divided by size of the L2 slice (Lines 1-3). WSS estimation approach is described in section 3.2. The number of cache line replacements increases with increase in cache requirement of an application. This happens when application shows a phase change. One additional L2 slice is allocated, if the number of cache line replacements per set is nonzero (Lines 4-7).

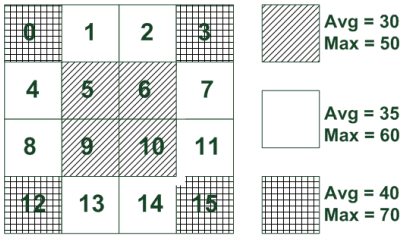


Figure 4: **average and maximum latency offered by each L2 slice to cores**

Communication costs are reduced by allocating L2 slices which offer lower average access latency(AAL). AAL of an L2 slice is calculated by taking average of latency offered by it to all cores. *averageLatency* list is created apriori by sorting all L2 slices in increasing order of their AALs. It is created as per network configuration, and is independent of an application. Hence, it is not computed in every monitoring period. By keeping *averageLatency* list constant, the number of remap table changes made in the consecutive monitoring periods is lowered, reducing the total number of clock cycles spent in the remap table changes. Our experimental results show that the remap table change overhead, in terms of clock cycles, is just on an average, 1.8% of the execution time. The average and maximum access latency offered by each L2 slice to cores on a 16 tiled CMP is shown in the Fig. 4 [3]. Higher preference is given to L2 slices in tiles 5, 6, 9 and 10 which offer lowest AAL of 30. Intuitively, they are located in the middle of the chip, so offer overall lower access latency to all cores.

If cache requirement is estimated as $numL2Slices$(number of L2 slices), then first $numL2Slices$ from the *averageLatency* list are allocated(Line 8). Rest of the L2 slices are mapped to these allocated L2 slices. Unallocated L2 slices are sorted in decreasing order of total number of accesses serviced by them. These slices are mapped to the allocated L2 slices ($numL2Slices$) so that the number of accesses serviced by each allocated slice is approximately equal. This ensures uniform distribution of cache accesses among the allocated L2 slices (Lines 9, 10).

The remap configuration change protocol can be implemented as a separate hardware module with no intervention of the operating system. The L2 controller from tile "0", as shown in Fig. 4, acts as a master. It receives the required statistics from the remaining L2 controllers over the NoC.

Cache lines are transferred to new location when mapping of an L1 slice is changed. To update the remap table in L1 controllers, the master uses *two-phase-commit* protocol. New L1 misses are put on-hold and after currently

[3]Please see technical report [4] for details on the tile link lengths and floor planning.

issued L1 misses are serviced, L2 controllers initiate cache line transfers. After all the cache line transfers are complete, the remap table is updated in L1 controllers and on-hold L1 misses are serviced. The active bit used to estimate WSS serves one more purpose. Cache lines with active bit reset are *not* considered to be part of the current working set and most likely will not be accessed in future. Such lines are invalidated and modified lines are written back to the memory. This avoids overhead of unnecessary cache line transfers. In the destination L2 slice, if invalid cache line is not available to receive the incoming line, then some other cache line is replaced depending on the cache replacement policy. *Our simulator takes into account power and time overhead caused due to cache line transfers and periodic monitoring events initiated by the remap master.*

3.4 Directory Assumption

Our base SNUCA architecture implements noninclusive cache. The directory information of a cache line is saved along with its tag, which includes a full bitmap of L1 sharers. For cache lines cached only in L1(s) and not in L2, a separate directory table is maintained to save L1 sharer(s) bitmap. We assume the directory table size is 5% of L2 slice. As a result, the directory table contains about 3000 entries. In remap, when an inclusive cache line is transferred, its directory information gets automatically transferred along with its tag to its new location. For noninclusive cache lines, its directory table entry is also transferred. If the directory table entries exceed 3000, then some other entry with least number of L1 sharers is invalidated. In summary, the remap policy can be applied to inclusive, noninclusive caches or caches with centralized directories.

3.5 Area Overhead and Scalability

L2 tag size increases as it contains cache lines from the remapped L2 slice besides its own. Hence the slice/tile identifier bits need to be included in the L2 tag. This does not add any additional access latency but size of the tag array increases by $log_2(\#tiles) * \#cachelines$. For example, in a CMP platform with 16 tiles, a physical address of 48 bits, an L2 cache per tile of 512KB in size, with 64B per cache line and an associativity of 16, the tag is 29 bits if remapping is not used. With remap, the tag increases to 33 bits. Additional space requirement for 512KB L2 tile is 8192[4] * (4+1) = 40K bits or 5KB, which is just 0.9% increase in the storage requirement. An additional bit is required to keep track of active lines in the current

[4]number of cache lines

monitoring period. Similarly, extra power required to perform the remap table lookup and additional tag comparison is minimal.

On doubling the number of tiles, tile ID bits size increases only by 1 bit. In the above example, storage overhead increases from 0.9% to 1.17% and 1.3%, if the number of tiles is increased from 16 to 32 and 64, respectively. For very large number of tiles (128 onwards), we believe, L2 cache will be shared and distributed among a subset of tiles and cache coherence will be maintained between these private L2 caches to lower L2 cache access latency. In that case, the remap policy can still be used within private L2 slices without much of an overhead.

4 Experimental Configuration

4.1 Applications used in Experiments

We evaluate multi-threaded workloads with one-to-one mapping between threads and cores (Table 2)[5]. This assumption is in line with other work done for CMP platforms. We have skipped initial serial portion and simulate only parallel section in all the test cases. We execute 1B instructions, unless otherwise mentioned. We test all workloads with 16 threads.

4.2 Experimental Setup And Methodology

We model all the system components with reasonable accuracy in our framework. We use SESC[13] to simulate a core, Ruby component from GEMS[14] to simulate the cache hierarchy and interconnects. DRAMSim is used to model the offchip DRAM. DRAMSim[15] uses MICRON [16] power model to estimate power consumed in DRAM accesses. Intacte [17] is used to estimate low level parameters of the interconnect such as the number of repeaters, wire width, wire length, degree of pipelining and power consumed by the interconnect. Power consumed by the cache components is estimated using CACTI 6.0.

In order to estimate the latency (in cycles) of a certain wire, we estimate area of all components in a tile and then create the floorplan which is shown in Fig. 5. We make following assumptions to determine area of various components at 32nm technology and 3GHz frequency:

- core : This is estimated based on the area of Intel Nehalem core[18]

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

Name	Description, WSS(L/M/S)
Alpbench Benchmark [12]	
MPGEnc	Encodes 15 Frames of size 640x336, M
MPGDec	Decodes 15 Frames of size 640x336, M
Splash2 Benchmark[2]	
Cholesky	blocked sparse matrix factorization on tk29, L
FFT	FFT on 1M points, M
LU (noncontinuous)	1024x1024 LU matrix factorization, S
Radix	Radix sort on 1M keys, M
FMM	simulate interaction of 16K bodies system, M
Water_spatial	simulation of 512 water molecules, M
Water_nsquared	simulation of 512 water molecules, M
Barnes	Barnes-Hut method on 16K bodies, M
Ocean (continuous)	512x512 grid points, L
PARSEC Benchmark[3]	
Blackscholes	SimLarge i/p, Financial Domain, S
Swaptions	SimLarge i/p, Financial Domain, M
Fluidanimate	SimMedium i/p, Animation, L
H.264 Encoder	SimLarge i/p, Media Domain, M

Table 2: Table shows applications used for study and their WSS information(L:Large, M:Medium, S:Small).

- cache : The L1 cache is of size 32KB whose area is very small and is included in the processor area. The area occupied by the L2 cache is obtained using CACTI 6.0. We assume directory information is stored along with each L2 slice. We conservatively assume area of per-tile directory to be negligible. If directory area is considered then interconnect lengths will increase which is more beneficial for the remap policy.

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

Fig. 5 also shows wire lengths and their power consumption. The latency of a link in clock cycles is equal to the number of its pipeline stages. To obtain power consumption of NoC, we compute the link activity and coupling factors of all links, caused due to the messages sent over NoC.

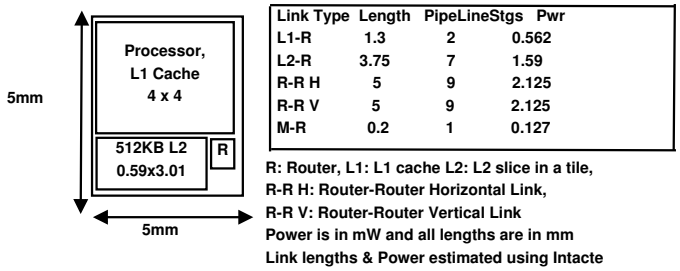


Figure 5: Floorplan of a tile with 512KB L2 slice.

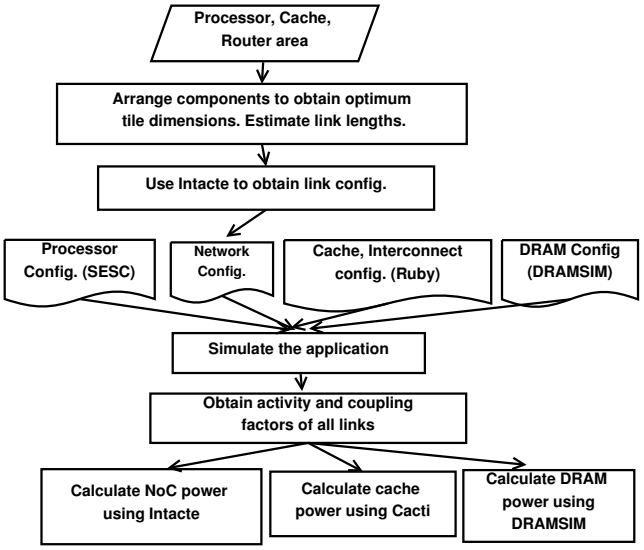


Figure 6: **Experimental Procedure**

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[19]), private, cache coherence using MOESI protocol
L2 Cache	512KB/tile, 16 way, 64B line size, 4 subbanks per slice, 3 cy. latency (estimated using CACTI), noninclusive, shared and distributed across all tiles
Directory	Tag bits of L2 cache line include full bitmap for L1 sharers. A separate table of 3000 entries maintains dir info. for cache lines not cached in L2 but only in L1s.
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
Off-chip DRAM	4GB, DDR2, 667MHz freq, 2 channels of 8B in width, 8 banks 16K rows, 1K columns, close page row management policy

Table 3: System configuration used in experiments

4.3 Simulation Procedure

Table 3 gives the system configuration used in our experiments. The flow chart in Fig. 6 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. Our simulator estimates the activity and coupling factors of all the links. Intacte determines power dissipated in NoC using these activity factors. Power consumed by the off-chip DRAM and on-chip cache is estimated using DRAMSim (MICRON) and CACTI power models, respectively.

5 Results

We compare the remap policy quantitatively with SNUCA, DNUCA, SNUCA.Drowsy and DNUCA.Drowsy (see section 2). We implement DNUCA as mentioned in [1]. ReMap.InfDir denotes the remap implementation with infinite directory table assumption, whereas, ReMap denotes remap implementation with directory table size as 5% of L2 slice size(see section 3.4).

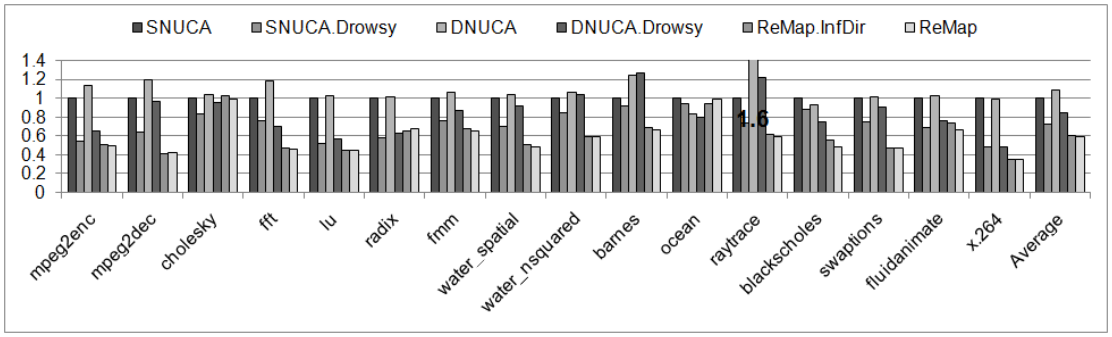


Figure 7: **Normalized Energy-Delay Product; Smaller EDP implies better program execution**

5.1 Energy-Delay Product (EDP)

Fig. 7 shows EDP savings obtained using various cache configurations. All readings are normalized wrt that of SNUCA. We consider energy dissipated in all memory subsystem components, viz., cache, NoC and DRAM while calculating EDP. In x.264, the remap policy achieves upto 64% EDP savings wrt SNUCA. Applications like mpgenc, mpgdec, x.264 have very small WSS. Hence, these applications show more than 50% EDP savings. On contrary, ocean has much larger WSS. Hence, the remap allocates on an average 13.9 L2 slices, giving negligible EDP savings. In ocean, however, drowsy technique gives 6% EDP savings in SNUCA and 20% in DNUCA. Higher EDP savings in DNUCA.Drowsy are because of lesser execution time in DNUCA wrt SNUCA. Similarly, drowsy technique gives higher savings in cholesky that the remap. Remap gives higher savings if WSS of an application is lesser than the available cache and drowsy gives higher savings if WSS is larger than the available cache. However, it should be noted that due to higher process variations seen in smaller transistors, reduced supply voltage decreases cache reliability [8, 7]. On an average, the remap gives 13% and 25% higher EDP savings over SNUCA.Drowsy and DNUCA.Drowsy.

Fig. 8 shows the average number of L2 slices allocated by the remap for each application. Applications like cholesky and ocean have large WSS, hence the remap allocates more L2 slices.

EDP savings obtained in case of the remap with infinite and finite sized directory table, for cache lines present only in L1, give approximately equal savings. This shows the directory table, maintained for noninclusive cache lines, with 5% size of L2 slice, is sufficient for the remap. Moreover, the remap is applicable to inclusive cache as well.

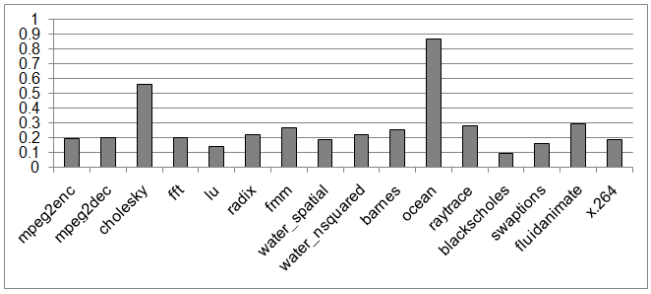


Figure 8: **Normalized average number of L2 slices allocated by the remap**

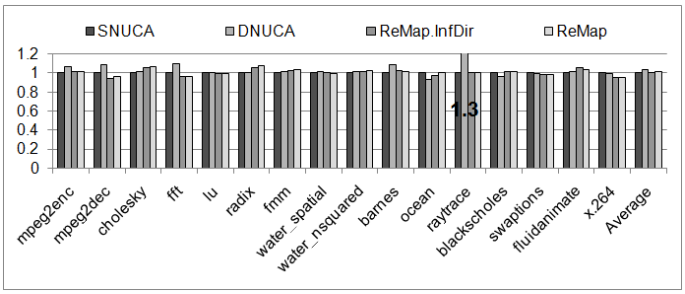


Figure 9: **Normalized execution time**

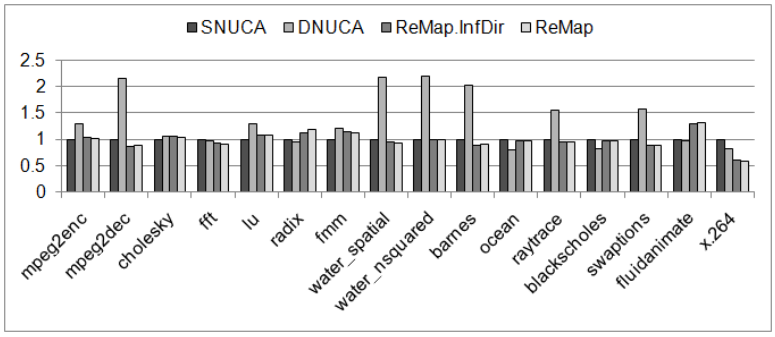


Figure 10: **Normalized time spent in transit**

5.2 Execution Time

In DNUCA, execution time of applications like blackscholes and ocean improves by 4% and 7% respectively (Fig. 9). This is because, in blackscholes, majority of data is shared between 2 threads. Hence, data is cached in nearer L2 slices. However, applications like mpegdec, fmm, raytrace show substan-

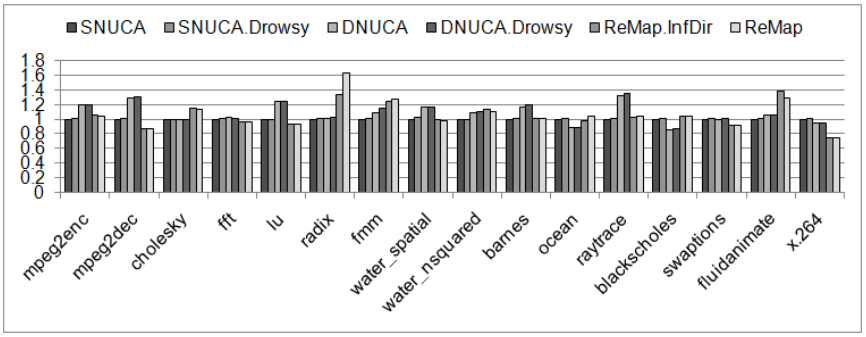
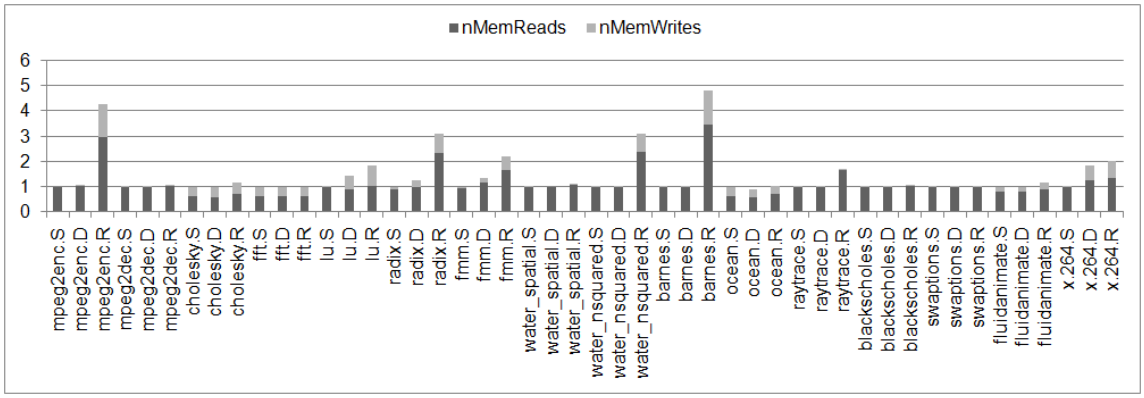


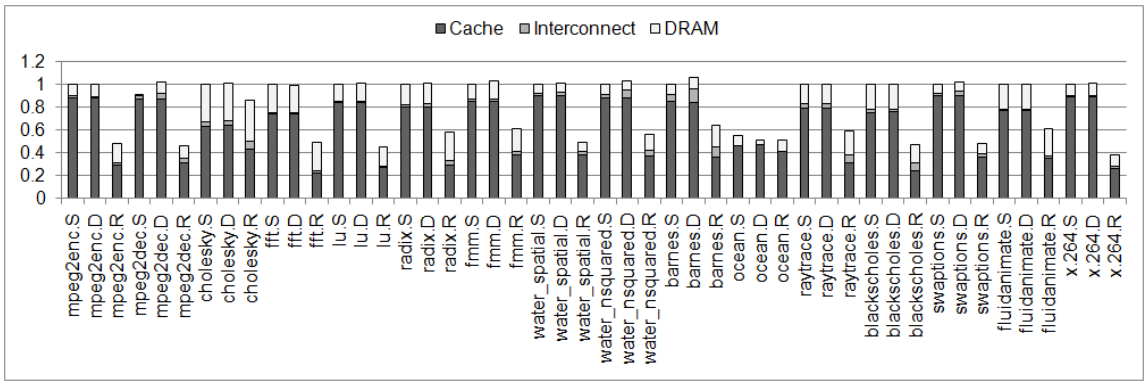
Figure 11: **Normalized L2 latency**

tial degradation in execution time with DNUCA compared to SNUCA as threads in these applications share a large amount of data. Hence, cache lines migrate in conflicting directions, increasing time spent in transit. Raytrace shows 25% degradation in execution time. This is mainly because on an L1 miss, all peer L2 slices have to be searched before data is read from offchip DRAM. This incurs additional transit time penalty. Fig. 10 shows time spent in transit by various cache configurations. Transit time is normalized wrt that in SNUCA. Clearly, applications like mpgdec, raytrace spent more time in transit in DNUCA. For applications like water_spatial and water_nsquared, even if more time is spent in transit in DNUCA, not proportionate degradation is seen in execution time. This is because degradation depends on the percentage of load/store instructions compared to other type of instructions. We simulate out-of-order type of core. Hence, L2 access latency gets overlapped by execution of other instructions. Our results invalidate the wide-spread belief that DNUCA performs better than SNUCA. Multi-threaded applications with more data sharing show significant degradation in DNUCA.

The remap improves execution time of x.264 and mpegdec by 6% and 4%, respectively. X.264 exhibits very poor *thread scalability*. It spawns only 3 concurrent threads even if executed with 16 threads as a command line parameter. As a result, the remap can allocate L2 slices nearer to these threads, showing overall 6% improvement in execution time. For other applications, remap gives large EDP savings with negligible or no performance degradation, except cholesky which shows maximum degradation of 7%. Fig. 11 compares normalized L2 access latency of all configurations. x.264, fft and mpgdec have lower latency than rest of the configurations. x.264 has 27% lower L2 latency. For rest of the applications like radix and cholesky, higher latency is due to write-backs done in remap during switch-off of L2 slice.



(a) Normalized number of memory reads and writes



(b) Normalized Power Consumption in cache, NoC and DRAM

Figure 12:

5.3 Number of Memory Accesses

As remap policy uses lesser cache than the reference SNUCA and DNUCA, it makes more number of memory accesses. Fig. 12(a) shows the number of memory accesses done by SNUCA, DNUCA and the remap policy. All are normalized wrt the total accesses made by SNUCA. Remap exhibits more number of memory accesses in case of mpegenc, radix, water_nsquared and barnes. However, power consumption in offchip DRAM does not increase proportionately as shown in Fig. 12(b). This is because on-chip cache operates at 3GHz, whereas, DRAM operates at much lower frequency of 667MHz, than cache. However, this might degrade execution time of an application. In radix, execution time degrades by 8% and in mpegenc and barnes negligible degradation of 1% is seen. However, as remap allocates only on an average 3.5 L2 slices out of 16 in radix, it shows 33% of EDP gains.

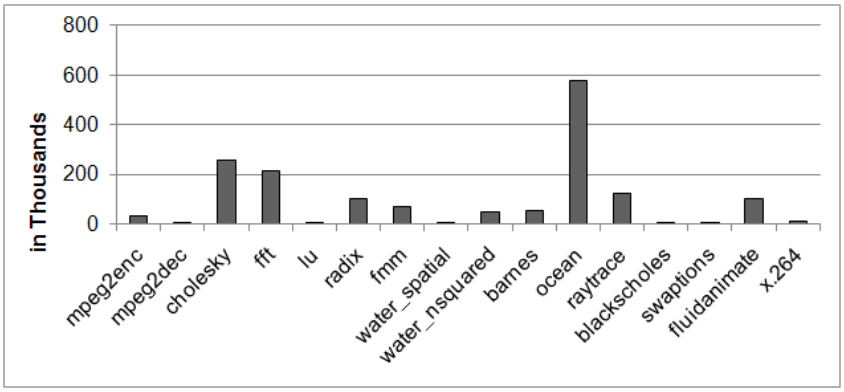


Figure 13: Remap policy overhead

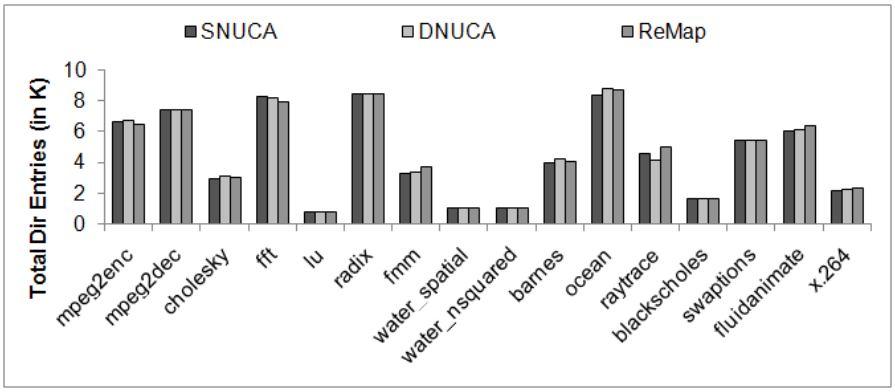
It should be noted that in case of ocean and cholesky which have large WSS, remap does not show increased number of memory accesses. It detects its WSS accurately and allocates almost all the sixteen slices. Thus execution time and EDP degradation is not seen in both of them. It gives EDP savings in all applications with small-to-large WSS.

5.4 Overhead in the Remap Policy

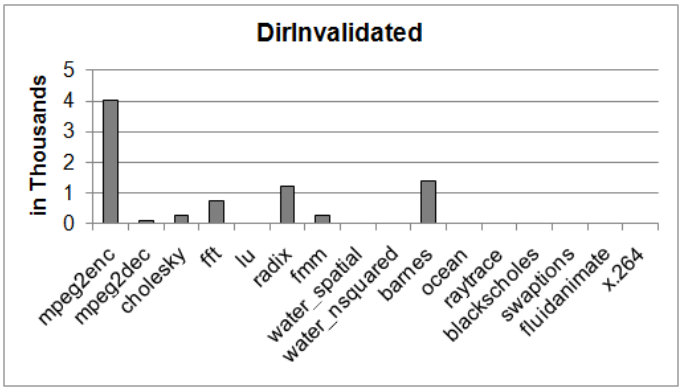
Fig. 13 also shows the total number cache lines transferred by the remap policy during simulation. On an average, it incurs overhead of 1.8%, in terms of the number of clock cycles required to transfer the cache lines.

5.5 Directory Table Invalidations

For inclusive caches, directory information which is a bitmap for L1 sharers, is included in the tag of each L2 cache line. In remap, we transfer directory information along with cache lines to their new location. Hence, no changes are required for inclusive caches/lines. In case of noninclusive caches, for cache lines which are present only in L1 cache and not included in L2 cache, a separate directory table is required for all cache policies, including our base SNUCA architecture. Fig. 14(a) shows the total number of directory table entries created by all policies. Please note that the separate directory table saves directory information only for cache lines which are cached in L1 and not in L2. As per Fig. 14(a), all cache policies create approximately equal number of directory table entries. However, in case of the remap, since we merge data from different slices in a single slice, we may create



(a) Total directory table entries created by various policies



(b) The number of directory table entries invalidated by the remap

more entries in a directory table than available entries. In that case, we replace some other directory entry with least number of L1 sharers. To replace a directory entry, we invalidate cache lines from its L1 sharers. We assume the size in KB of the directory table as 5% of L2 slice. With this assumption, the directory contains 3000 entries. If the number of entries exceed 3000, then some other directory table entry is replaced. Fig. 14(b) shows the total number of directory entries invalidated by the remap. As we accurately predict and allocate required amount of L2 slices, the number of directory entries invalidated, are negligible. As shown in Fig. 7, EDP savings obtained with infinite and finite size directory table do not vary much. As mentioned earlier, remap can be applied to inclusive or noninclusive caches with centralized or distributed directories.

5.6 Sensitivity to Monitoring Period

We use 2 million clock cycle as the monitoring period. As mentioned in Section 3.2, all applications that we study, have reuse time less than 2M cycles. It means that if the address is not used in last 2M cycles then it will not be used in next 2M clock cycles as well. Hence, we determine WSS of an application after every 2M clock cycles. Fig. 14 compares EDP gains obtained using 1.5M, 2M and 2.5M clock cycles as the monitoring period. On varying monitoring period, EDP gains show negligible variation[4].

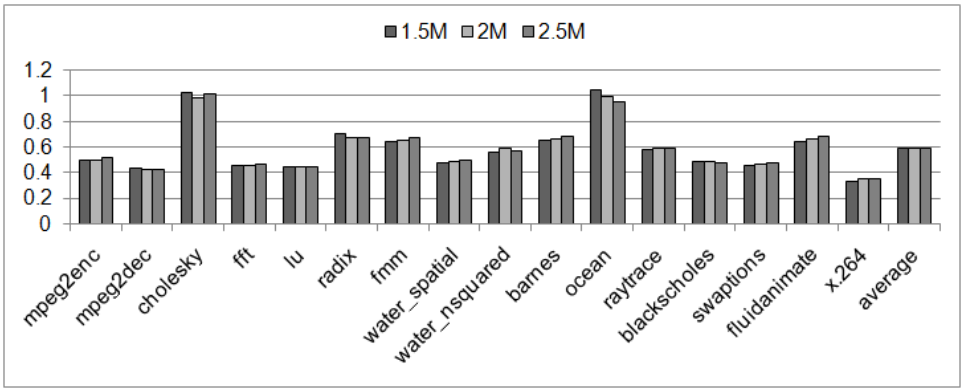


Figure 14: EDP gains obtained using the remap policy show minimal variation on varying monitoring period

6 Conclusions and Future Work

Due to increased on-chip cache size, its leakage power consumption has become the major component of the memory subsystem power consumption. Hence, we propose to switch off farther L2 slices in an over-allocated cache to save leakage power and map those slices onto nearer L2 slices using the remap policy. Apart from reducing leakage power consumption, the remap policy also shows up-to 6% improvement in execution time. On an average, the remap policy achieves 41% and 49% EDP savings wrt SNUCA and DNUCA. Unlike drowsy cache technique, the remap policy is not prone to the transient errors.

Bardine et al.[10] proposed to vary associativity of DNUCA cache to save leakage power. They primarily consider DNUCA cache with single and dual core platform in which cores are on one side of NoC and DNUCA cache is on the other side of NoC. They use ratio of the number of hits to the farthest

cache line to the nearest line in a set to estimate the required cache associativity. Their heuristic cannot be applied to a scalable tiled architecture. This is because, in tiled architecture, cache lines in a set, which are nearer to one core can be far for other cores, unlike a single or dual core scenario. Hence, in future, we plan to explore a scalable heuristic to vary cache associativity on a scalable tiled architecture.

References

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

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

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

[4] A. Mandke, Y. N. Srikant, and A. Bharadwaj, "Adaptive power optimization of onchip snuca cache on tiled chip multicore platform using remap policy," no. IISc-CSA-TR-2011-02. [Online]. Available: {http://www.csa.iisc.ernet.in/TR/2011/2/}

[5] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *ISCA*, 2002.

[6] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *ISCA*, 2001.

[7] G. Gammie, A. Wang, H. Mair, R. Lagerquist, R. Philippe, G. Sumanth, and K. Uming, "Smartreflex power and performance management technologies for 90nm, 65nm, and 45nm mobile application processors," in *Proc. of IEEE Journal*, Feb, 2010.

[8] B. Calhoun and A. Chandrakasan, "Static noise margin variation for sub-threshold sram 65-nm CMOS," in *IEEE Journal on Solid State Circuits*, 2006.

[9] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *MICRO*, 1999.

[10] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, and C. A. Prete, "Way adaptable DNUCA caches," *Int. J. High Perform. Syst. Archit.*, 2010.

[11] A. Mandke, A. Bharadwaj, and Y. N. Srikant, "Applying Genetic Algorithms to Optimize Power in Tiled SNUCA Chip Multicore Architectures," in *ACM SAC*, 2011.

[12] 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.

[13] 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.

[14] 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, "Multifacets general execution-driven multiprocessor simulator (gems) toolset," 2005.

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

[16] "Micron DRAM power data sheet." [Online]. Available: http://www.micron.com/products/partdetail?part=MT47H128M8HQ-3E

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

[18] "Intel Nehalem." [Online]. Available: http://www.3dnow.net/phpBB2/viewtopic.php?f=1\&t=1474\&p=6720

[19] 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