Optimizing Communication and Capacity in a 3D Stacked Reconfigurable Cache Hierarchy [*]

Niti Madan[†], Li Zhao[‡], Naveen Muralimanohar[†], Aniruddha Udipi[†],
Rajeev Balasubramonian[†], Ravishankar Iyer[‡], Srihari Makineni[‡], Donald Newell[‡]
[†] School of Computing, University of Utah
[‡] System Technology Lab, Intel Corporation

Abstract

Cache hierarchies in future many-core processors are expected to grow in size and contribute a large fraction of overall processor power and performance. In this paper, we postulate a 3D chip design that stacks SRAM and DRAM upon processing cores and employs OS-based page coloring to minimize horizontal communication of cache data. We then propose a heterogeneous reconfigurable cache design that takes advantage of the high density of DRAM and the superior power/delay characteristics of SRAM to efficiently meet the working set demands of each individual core. Finally, we analyze the communication patterns for such a processor and show that a tree topology is an ideal fit that significantly reduces the power and latency requirements of the on-chip network. The above proposals are synergistic: each proposal is made more compelling because of its combination with the other innovations described in this paper. The proposed reconfigurable cache model improves performance by up to 19% along with 48% savings in network power.

Keywords: *multi-core processors, cache and memory hierarchy, non-uniform cache architecture (NUCA), page coloring, on-chip networks, SRAM/DRAM cache reconfiguration.*

1. Introduction

The design of cache hierarchies for multi-core chips has received much attention in recent years (for example, [5, 8, 10, 18, 38, 48]). As process technologies continue to shrink, a single chip will accommodate many mega-bytes of cache storage and numerous processing cores. It is well known that the interconnects that exchange data between caches and cores represent a major bottleneck with regard to both power and performance. Modern Intel chips already accommodate up to 27 MB of cache space [29]; interconnects have been attributed as much as 50% of total chip dynamic power [28]; on-chip networks for large tiled chips have been shown to consume as much as 36% of total chip power [23, 44]; long on-chip wires and router pipeline delays can lead to cache access times of many tens of cycles [12, 36]. Not only will we require intelligent mechanisms to allocate cache space among cores, we will also have to optimize the interconnect that exchanges data between caches and cores. This paper makes an attempt at addressing both of these issues.

3D stacking of dies has been demonstrated as a feasible technology [46] and is already being commercially employed in some embedded domains [35, 39]. In most commercial examples, 3D is employed to stack DRAM memory upon CPU cores [35, 39]. This is especially compelling because future multi-cores will make higher memory bandwidth demands and the inter-die interconnect in a 3D chip can support large data bandwidths. Early projections for Intel's Polaris 80-core prototype allude to the use of such 3D stacking of DRAM to feed data to the 80 cores [42]. Given the commercial viability of this technology, a few research groups have already begun to explore the architectural ramifications of being able to stack storage upon CPUs [21, 24–27].

In this paper, we first postulate a physical processor design that is consistent with the above trends. We then take advantage of the following three key innovations to architect a cache hierarchy that greatly reduces latency and power: (i) we employ page coloring to localize data and computation, (ii) we propose the use of cache reconfiguration to accommodate large working sets for some cores, (iii) we identify a network topology that best matches the needs of data traffic and incurs low delay and power overheads.

The proposed processor employs three dies stacked upon each other (see Figure 1). The lowest die contains the processing cores (along with the corresponding L1 caches). The second die is composed entirely of SRAM cache banks (forming a large shared L2 cache) and employs an on-chip network so that requests from the CPU can be routed to the correct bank. The third die is composed of DRAM banks that serve to augment the L2 cache space provided by the second SRAM die. It is also possible to stack many more DRAM dies upon these three dies to implement main memory [26], but we regard this as an orthogonal design choice and do not consider it further in this paper.

The L2 cache banks are organized as a non-uniform

[*]This work was supported in parts by NSF grants CCF-0430063, CCF-0811249, NSF CAREER award CCF-0545959, Intel, and the University of Utah.

cache architecture (NUCA) [22]. The request from the processing core is transmitted to the SRAM die through inter-die vias. From here, the request is propagated to the appropriate bank through the on-chip network and the latency for the access is a function of the proximity of this bank to the processing core. Many recent papers have explored various mechanisms to reduce average access times in a NUCA cache [5, 7, 9, 10, 18, 48]. Most dynamic (D-NUCA) mechanisms can cause data to be placed anywhere on the chip, requiring search mechanisms to locate the data. We dis-regard these algorithms because of the complexity/cost of search mechanisms and resort to a static-NUCA (S-NUCA) organization, where a given physical address maps to a unique bank in the cache (the physical address maps to a unique bank and set within that bank; the ways of the set may be distributed over multiple subarrays within that bank). To improve the proximity of storage and computation, we employ page coloring to ensure that data is placed in "optimal" banks. The idea of employing page coloring to dictate data placement in a NUCA cache was proposed in a recent paper by Cho and Jin [11]. Ideally, a low-overhead run-time mechanism would be required to estimate the usage of a page so that pages can be dynamically migrated to their optimal locations [1]. The design of such mechanisms is a non-trivial problem in itself and beyond the scope of this paper. For this work, we carry out an off-line analysis to identify pages that are private to each core and that are shared by multiple cores. Private pages are placed in the bank directly above the core and shared pages are placed in one of four central banks.

With the above page coloring mechanism in place, we expect high data locality and most cores will end up finding their data in the L2 cache bank directly above. In a multi-programmed workload, each core may place different demands on the L2 cache bank directly above. In a multi-threaded workload, the centrally located cache banks will experience higher pressure because they must accommodate shared data in addition to the data that is private to the corresponding cores. These varied demands on each cache bank can perhaps be handled by allowing a core to spill some of its pages into the adjacent banks. However, this not only increases the average latency to access that page, it also places a higher bandwidth demand on the inter-bank network, a trait we are striving to avoid (more on this in the next paragraph). Hence, we instead spill additional pages into the third dimension – to the DRAM bank directly above the SRAM cache bank. Note that the DRAM bank and SRAM bank form a single large vertical slice in the same L2 cache level. When the DRAM space is not employed, each SRAM cache bank accommodates 1 MB of cache space. If this space is exceeded, the DRAM bank directly above is activated. Since DRAM density is eight times SRAM density, this allows the cache space to increase to roughly 9 MB. While DRAM has poorer latency and power characteristics than SRAM, its higher density allows a dramatic increase in cache space without the need for many more stacked dies (that in turn can worsen temperature characteristics). Further, we architect the combined SRAM-DRAM cache space in a manner that allows non-uniform latencies and attempts to service more requests from the faster SRAM die. DRAM bank access itself has much lower cost than traditional DRAM main memory access because the DRAM die is partitioned into many small banks and a single small DRAM bank is looked up at a time without traversing long wires on the DRAM die. This results in a heterogeneous reconfigurable cache space, an artifact made possible by 3D die stacking. A reasonable alternative would be the implementation of a 3-level cache hierarchy with the top DRAM die serving as an independent L3 cache. A static design choice like that would possibly complicate cache coherence implementations, incur the latency to go through three levels for many accesses, and reduce the effective capacity of the L2 because of the need for L3 tags. We believe that the overall design is made simpler and faster by growing the size of the L2 cache bank on a need basis.

Finally, we examine the traffic patterns generated by the cache hierarchy described above. Most requests are serviced by the local SRAM and DRAM cache banks and do not require long traversals on horizontal wires. Requests to shared pages are directed towards the centrally located banks. Requests are rarely sent to non-local non-central banks. Such a traffic pattern is an excellent fit for a tree network (illustrated in Figure 1). A tree network employs much fewer routers and links than the grid network typically employed in such settings. Routers and links are cumbersome structures and are known to consume large amounts of power and area [23, 44] – hence, a reduction in routers and links has many favorable implications. Tree networks perform very poorly with random traffic patterns, but the use of intelligent page coloring ensures that the traffic pattern is not random and best fits the network topology. A tree network will likely work very poorly for previously proposed D-NUCA mechanisms that can place data in one of many possible banks and that require search mechanisms. A tree network will also work poorly if highly pressured cache banks spill data into neighboring banks, making such a topology especially apt for the proposed design that spills data into the third dimension.

The contributions of the paper are:

- A synergistic combination of page coloring, cache reconfiguration, and on-chip network design that improves performance by up to 62%.
- The design of a heterogeneous reconfigurable cache and policies to switch between configurations.

The paper is organized as follows. Section 2 provides basic background on recent innovations in multi-core cache design and related work. Section 3 describes our proposed cache architecture. Results are discussed in Section 4 and we draw conclusions in Section 5.

2. Background and Related Work

Most future large caches are expected to have NUCA architectures [22]. A large shared L2 or L3 cache can either be placed in a contiguous region or split into slices and associated with each core (tile). Early designs split the ways of a set across multiple banks, allowing a given block to have multiple possible residences. Policies were proposed to allow a block to gravitate towards a way/bank that minimized access time (D-NUCA [22]). However, this led to a non-trivial search problem: a request had to look in multiple banks to eventually locate data. A static-NUCA (S-NUCA) design instead places all ways of a set in a single bank and distributes sets across banks. Given a block address, the request is sent to a unique bank, that may or may not be in close proximity. In a recent paper, Cho and Jin [11] show that intelligent page coloring can influence address index bits so that the block is mapped to a set and bank that optimizes access time. The work in this paper is built upon the page coloring concept to improve access times and eliminate search. Other papers that attempt to improve data placement with a D-NUCA approach include [5, 7, 9, 10, 48]. A number of papers also attempt to improve multi-core cache organizations by managing data cooperatively within a collection of private caches [8, 15, 38]. In these papers, if a core's private cache cannot provide the necessary capacity, blocks are spilled into the private caches of neighboring cores.

Recent papers have also proposed innovative networks for large caches. Jin et al. [20] propose a halo network that best meets the needs of a single-core D-NUCA cache, where requests begin at a cache controller and radiate away as they perform the search. Beckmann and Wood [6] propose the use of transmission lines to support low-latency access to distant cache banks. Muralimanohar and Balasubramonian [31] propose a hybrid network with different wiring and topologies for the address and data networks to improve access times. Guz et al. [17] propose the Nahalal organization to better manage shared and private data between cores.

A number of recent papers have proposed cache hierarchy organizations in 3D. Li et al. [24] describe the network structures required for the efficient layout of a collection of cores and NUCA cache banks in 3D. Some bodies of work implement entire SRAM cache structures on separate dies [21, 25, 27]. Loh [26] proposes the changes required to the memory controller and DRAM architecture if several DRAM dies are stacked upon the CPU die to implement main memory. Ours is the first body of work that proposes reconfiguration across dies and combines heterogeneous technologies within a single level of cache.

Prior work on reconfigurable caches has been restricted to a single 2D die and to relatively small caches [3, 34, 47]. Some prior work [19, 33, 43] logically splits large cache capacity across cores at run-time and can be viewed as a form of reconfiguration.

3. Proposed Ideas

3.1. Proposed NUCA Organization

We first describe the basic model for access to the L2 cache in our proposed implementation. As shown in Figure 1, the bottom die contains 16 cores. The proposed ideas, including the tree topology for the on-chip network, will apply to larger systems as well. We assume 3.5 mm^2 area for each core at 32 nm technology, based on a scaled version of Sun's Rock core [41]. Each die is assumed to have an area of around 60 mm^2. Each core has its own private L1 data and instruction caches. An L1 miss causes a request to be sent to the L2 cache implemented on the dies above. The SRAM die placed directly upon the processing die is partitioned into 16 banks (we will later describe the role of the DRAM die). Based on estimates from CACTI 6.0 [32], a 1 MB SRAM cache bank and its associated router/controller have an area roughly equal to the area of one core. Each bank may itself be partitioned into multiple subarrays (as estimated by CACTI 6.0) to reduce latency and power. Each bank is associated with a small cache controller unit and a routing unit. On an L1 miss, the core sends the request to the cache controller unit directly above through an inter-die via pillar. Studies [16, **?**] have shown that high bandwidth vias can be implemented and these vias have pitch values as low as 4 μm [16].

The L2 cache is organized as a static-NUCA. Four bits of the physical address are used to map a data block to one of the 16 banks. As a result, no search mechanism is required – the physical address directly specifies the bank that the request must be routed to. Once an L2 cache controller receives a request from the core directly below, it examines these four bits and places the request on the on-chip network if destined for a remote bank. Once the request arrives at the destination bank, the cache subarrays in that bank are accessed (more details shortly) and data is returned to the requesting core by following the same path in the opposite direction. The L2 tag maintains a directory to ensure coherence among L1 caches and this coherence-related traffic is also sent on the on-chip network.

It must be noted that the baseline model described so far is very similar to tiled multi-core architectures that are commonly assumed in many papers (for example, [48]). These are typically referred to as logically shared, but physically distributed L2 caches, where a slice of L2 cache is included in each tile. The key difference in our model is that this slice of L2 is separated into a second SRAM die.

3.2. Page Coloring

An S-NUCA organization by itself does not guarantee low access times for L2 requests. Low access times can be obtained by ensuring that the data requested by a core is often placed in the cache bank directly above. Page coloring is a well-established OS technique that exercises greater

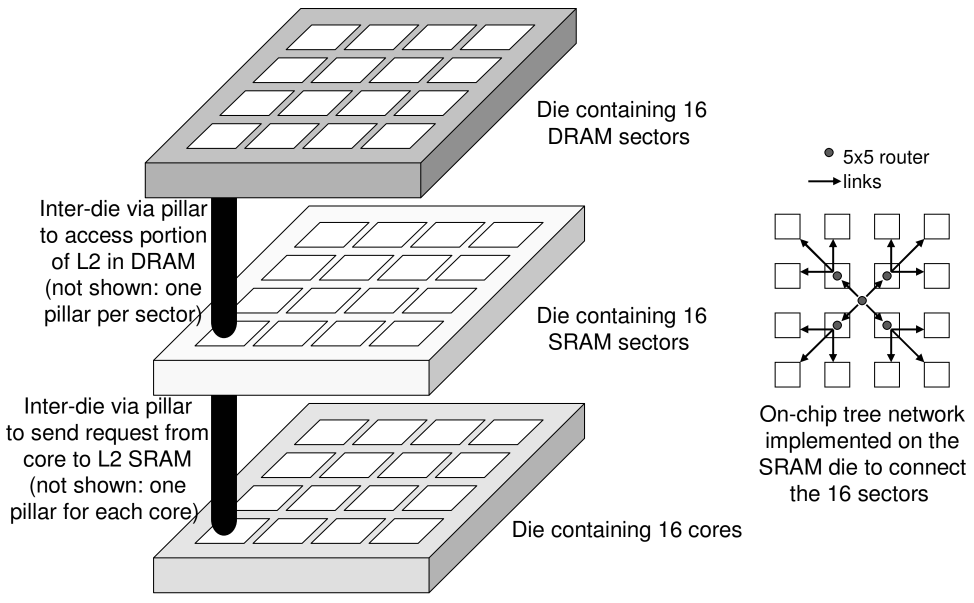

Die containing 16 DRAM sectors
5x5 router
links
Inter-die via pillar to access portion of L2 in DRAM (not shown: one pillar per sector)
Die containing 16 SRAM sectors
Inter-die via pillar to send request from core to L2 SRAM (not shown: one pillar for each core)
On-chip tree network implemented on the SRAM die to connect the 16 sectors
Die containing 16 cores

Figure 1. **Stacked 3D processor layout. The bottom die contains the 16 cores, the second die contains 16 SRAM banks, and the top die contains 16 DRAM banks. Inter-die via pillars (one for each bank) are used to implement vertical connections between cores and DRAM/SRAM banks. Horizontal communication happens on the on-chip tree network (shown on the right) implemented on the second die. There are no horizontal connections between cores (banks) on the bottom (top) die.**

control on the values assigned to bits of the physical address. Traditionally, page coloring has been employed to eliminate the aliasing problem in large virtually indexed caches. In the context of an S-NUCA cache, page coloring can be employed to influence the four bits of the physical address that determine the bank within the NUCA cache. If the entire L2 cache size is 16 MB (each bank is 1 MB) and is 4-way set-associative with 64 byte line size, a 64-bit physical address has the following components: 6 bits of offset, 16 bits of set index, and 42 bits of tag. If the page size is 4 KB, the 12 least significant bits are the page offset and the 52 most significant bits are the physical page number. The four most significant bits of the set index are also part of the physical page number. These four bits can be used to determine the bank number. Since the OS has control over the physical page number, it also has control over the bank number.

For pages that are accessed by only a single core, it is straightforward to color that page such that it maps to the bank directly above the core. For pages that are accessed by multiple cores, the page must ideally be placed in a bank that represents the center-of-gravity of all requests for that page. Creating such a mapping may require page migration mechanisms and hardware counters to monitor a page's access pattern and cache bank pressure. These are non-trivial policies that are beyond the scope of this work. For now, we assume that mechanisms can be developed to closely match the page allocation as computed by an off-line oracle analysis. In other words, the optimal cache bank for a page is pre-computed based on the center-of-gravity of requests for that page from various cores. As a result, shared data in multi-threaded workloads tend to be placed in the four banks in the center of the chip. We also devise policies to map a shared instruction (code) page either in the central shared bank or as replicated read-only pages in each bank. We evaluate the following three (oracle) page coloring schemes (also shown in Figure 2). Note that in all these schemes, private pages are always placed in the bank directly above; the differences are in how shared data and instruction pages are handled.

- **Share4:D+I :** In this scheme, we employ a policy that assigns both shared data and instruction pages to the central four banks. The shared bank is selected based on the proximity to the core that has maximum accesses to that page. If the program has a high degree of sharing, then we expect the central four banks to have increased pressure. This may cause the central banks to enable their additional DRAM cache space.
- **Rp:I + Share4:D :** In this scheme, we color all shared data pages so they are mapped to central banks. We replicate shared instruction pages and assign them to each accessing core. This causes the bank pressure to increase slightly for all private banks as the working set size of instruction pages is typically very small. This improves performance greatly as code pages are frequently accessed in the last-level cache (LLC) in commercial workloads.
- **Share16:D+I :** In order to uniformly utilize the available cache capacity, we color all shared pages (data and code) and distribute them evenly across all 16 banks. This page coloring scheme does not optimize

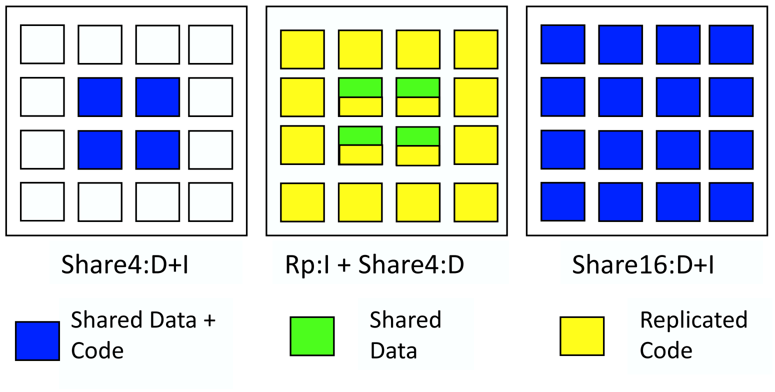


Figure 2. **Page Coloring Schemes**

for latency for shared pages, but attempts to maximize SRAM bank hit rates.

Note that the above schemes may require minimal OS and hardware support. These schemes rely upon detection of shared versus private pages and code versus data pages.

3.3. Reconfigurable SRAM/DRAM Cache

As described, the L2 cache is composed of 16 banks and organized as an S-NUCA. Each bank in the SRAM die has a capacity of 1 MB. If the threads in a core have large working sets, a capacity of 1 MB may result in a large miss rate. Further, centrally located banks must not only service the local needs of their corresponding cores, they must also accommodate shared pages. Therefore, the pressures on each individual bank may be very different. One possibility is to monitor bank miss rates and re-color some pages so they spill into neighboring banks and release the pressure in "popular" banks. Instead, we propose a reconfigurable cache that takes advantage of 3D stacking to seamlessly grow the size of a bank in the vertical dimension. This allows the size of each bank to grow independently without impacting the capacity or access time of neighboring banks. Many proposals of 2D reconfigurable caches already exist in the literature: they allow low access times for small cache sizes but provide the flexibility to incorporate larger capacities at longer access times. The use of 3D and NUCA makes the design of a reconfigurable cache especially attractive: (i) the spare capacity on the third die does not intrude with the layout of the second die, nor does it steal capacity from other neighboring caches (as is commonly done in 2D reconfigurable caches [3, 47]), (ii) since the cache is already partitioned into NUCA banks, the introduction of additional banks and delays does not greatly complicate the control logic, (iii) the use of a third dimension allows access time to grow less than linearly with capacity (another disadvantage of a 2D reconfigurable cache).

While growing the size of a bank, the third die can implement SRAM subarrays identical to the second die, thus allowing the bank size to grow by a factor of two. But reconfiguration over homogeneous banks does not make much sense: the added capacity comes at a trivial latency cost, so there is little motivation to employ the smaller cache configuration. Reconfiguration does make sense when employing heterogeneous technologies. In this case, we advocate the use of a third die that implements DRAM banks, thus allowing the bank size to grow by up to a factor of nine[1]. By stacking a single DRAM die (instead of eight SRAM dies), we provide high capacity while limiting the growth in temperature and cost, and improving yield. Since DRAM accesses are less efficient than SRAM accesses, dynamic reconfiguration allows us to avoid DRAM accesses when dealing with small working sets.

Organizing Tags and Data.

The SRAM bank is organized into three memory arrays: a tag array, a data array, and an adaptive array that can act as both tag and data arrays. As with most large L2s, the tags are first looked up and after the appropriate way is identified, the data subarray is accessed. Assuming that each block is 64 bytes and has a 32 bit tag (including directory state and assuming a 36-bit physical address space), the corresponding tag storage is 64 KB. The use of DRAM enables the total bank data storage to grow to 9 MB, requiring a total 576 KB of tag storage. Since tags are accessed before data and since SRAM accesses are cheaper than DRAM accesses, it is beneficial to implement the tags in SRAM. As a result, 512 KB of SRAM storage that previously served as data subarray must now serve as tag subarray. Compared to a typical data array, a tag array has additional logic to perform tag comparison. The data sub-array has more H-tree bandwidth for data transfer compared to its tag counterpart. To keep the reconfiguration simple, the proposed adaptive array encompasses the functionalities of both tag and data array.

Growing Associativity, Sets, Block-Size.

The increased capacity provided by DRAM can manifest in three forms (and combinations thereof): (i) increased associativity, (ii) increased number of sets, and (iii) increased block size.

[1]Published reports claim a factor of eight difference in the densities of SRAM and DRAM [40].

Configuration	Access time (ns)	Total Energy per access(nJ)	Area (mm^2)
Baseline SRAM 1 MB bank	3.13	0.699	2.07
Reconfigurable cache (ways) DRAM	6.71	1.4	3.23
Reconfigurable cache (sets) DRAM	6.52	1.36	2.76
Reconfigurable cache (block size) DRAM	5.43	51.19	1.44

Table 1. **Access time, energy, and area for various cache configurations at a 4 GHz clock derived from [32].**

The first form of reconfiguration allows the bank to go from 4-way to 34-way set associative. 32 data ways are implemented on the DRAM die and two of the original four data ways remain on the SRAM die after half the data subarrays are converted to tag subarrays. Such an approach has the following advantages: (i) dirty lines in the two ways in the SRAM die need not be flushed upon every reconfiguration, (ii) every set in the bank can have two of its ways in relatively close proximity in SRAM. The primary disadvantage is the power and complexity overhead of implementing 34-way set-associativity. We can optimize the cache lookup further by moving the MRU block into the two SRAM ways on every access in the hope that this will reduce average access times. With this policy, a hit in DRAM space will require an SRAM and DRAM read and write. In our evaluations, we employ reconfiguration of the number of ways, but do not attempt the optimization where MRU blocks are moved into SRAM.

The second form of reconfiguration causes an increase in the number of sets from 2K to 16K (we will restrict ourselves to power-of-two number of sets, possibly leading to extra white space on the top die). When cache size is increased, nearly every dirty cache line will have to be flushed. When cache size is decreased, lines residing in the SRAM data subarrays need not be flushed. The large cache organization has a more favorable access time/power for a fraction of the sets that map to SRAM data subarrays. The page coloring mechanism could attempt to color critical pages so they reside in the sets that map to SRAM.

The third form of reconfiguration increases the block size from 64 bytes to 512 bytes (again, possibly resulting in white space). Note that this approach does not increase the tag space requirement, so 1 MB of data can be placed in SRAM, while 7 MB is placed in DRAM. This has the obvious disadvantage of placing higher pressure on the bus to main memory and also higher energy consumption for accesses. While re-configuring the number of sets or block size, care must be taken to not change the address bits used to determine the bank number for an address.

Thus, there are multiple mechanisms to reconfigure the cache. The differences are expected to be minor unless the application is highly sensitive to capacity (8 MB versus 8.5 MB) and memory bandwidth. While some mechanisms can escape flushing the entire cache, these savings are relatively minor if cache reconfiguration is performed infrequently. In our evaluations, we only focus on the first reconfiguration approach that changes the number of ways.

It is worth noting that banks in the DRAM die are laid out very similar to banks in the SRAM die. Our estimates for DRAM delay, power, and area are based on CACTI-6.0 and discussions with industrial teams. Table 1 summarizes the delay, power, and area of the considered organizations. The DRAM banks can also be statically employed as a level-3 cache. However, this would significantly reduce the size of the SRAM L2 cache as 0.5 MB space on the second die would have to be designated as L3 tags. This may have a negative impact for several applications with moderate working-set sizes (not evaluated in this paper). It will also increase latencies for L3 and memory accesses because tags in multiple levels have to be sequentially navigated.

Having described the specific implementation, the following additional advantages over prior 2D designs [3, 47] are made clear: (i) a dramatic 8x increase in capacity is possible at a minor delay overhead, (ii) only two configurations are possible, enabling a simpler reconfiguration policy, (iii) each cache bank can reconfigure independently, thus avoiding a flush of the entire L2 all at one time.

Reconfiguration Policies.

We next examine the design of a reconfiguration policy. The frequency of reconfiguration is a function of the overheads of a cache flush and cache warm-up. Up to 16K cache lines will have to be flushed or brought in upon every reconfiguration. While the fetch of new lines can be overlapped with execution, cache flush will likely have to stall all requests to that bank. A state-of-the-art memory system can handle a peak throughput of 10 GB/s [36]. A complete flush will require a stall of roughly 100 K cycles. For this overhead to be minor, a reconfiguration is considered once every 10 M cycles. Reconfiguration policies are well-studied (for example, [2, 4, 13, 14]). We design two simple policies that are heavily influenced by this prior work.

Every 10 M cycles, we examine the following two metrics for each cache bank: bank miss rate and usage. A high bank miss rate indicates the need to grow cache size, while low usage indicates the need to shrink cache size. It is important to pick a low enough threshold for usage, else the configurations can oscillate. Usage can be determined by maintaining a bit per cache block that is set upon access and re-set at the start of every 10 M cycle interval (this is assumed in our simulations). There also exist other complexity-effective mechanisms to estimate usage [33].

In an alternative policy, various statistics can be maintained per bank to determine if the application has moved to a new phase (a part of application execution with different behavior and characteristics). If a substantial change in these statistics is detected over successive large time intervals (*epochs*), a phase chase is signaled. Upon a phase change, we simply implement each of the two configuration choices and measure instruction throughput to determine which configuration is better (referred to as *exploration*). Each exploration step has to be long enough to amortize cache warm-up effects. The optimal organization is employed until the next phase change is signaled. If phase changes are frequent, the epoch length is doubled in an attempt to capture application behavior at a coarser granularity and minimize the overheads of exploration.

Since these policies are strongly based on prior work (most notably [4]), and have been shown to be effective in other domains, we don't focus our efforts on evaluating the relative merits of each of these policies. Because of the large sizes of the caches and epochs, extremely long simulations will be required to observe any interesting artifacts with regard to program phases. Our simulations model the first reconfiguration policy that chooses to grow or shrink cache size based on miss rate and usage, respectively. In practice, we believe that the second reconfiguration policy may be more effective because it avoids the overheads of having to keep track of cache line usage.

3.4. Interconnect Design

Most papers on NUCA designs or tiled multi-cores have employed grid topologies for the inter-bank network. Grid topologies provide high performance under heavy load and random traffic patterns. This was indeed the case for prior D-NUCA proposals where data could be placed in one of many possible banks and complex search mechanisms were required to locate data. However, with the 3D reconfigurable cache hierarchy and the use of S-NUCA combined with page coloring, the traffic patterns are typically very predictable. For multi-programmed workloads, most requests will be serviced without long horizontal transfers and for multi-threaded workloads, a number of requests will also be serviced by the four central banks. There will be almost no requests made to non-local and non-central banks. With such a traffic pattern, a grid topology is clearly overkill. Many studies have shown that on-chip routers are bulky units. They not only consume a non-trivial amount of area and power [23], commercial implementations also incur delay overheads of four [23] to eight [30] cycles. Hence, it is necessary that we find a minimal topology that can support the required traffic demands. Given the nature of the traffic pattern, where most horizontal transfers radiate in/out of the central banks, we propose the use of a tree topology, as shown in Figure 1. This allows the use of four 5x5 routers in the central banks and one 4x4 router as the root. In addition, the 12 leaf banks need buffers for incoming flits and some control logic to handle flow control for the outgoing flits. Note that this network is only employed on the second die – there are no horizontal links between banks on the first and third dies.

4. Results

4.1. Methodology

We use a trace-driven platform simulator ManySim [49] for our performance simulations. ManySim simulates the platform resources with high accuracy, but abstracts the core to optimize for speed. The core is represented by a sequence of compute events (collected from a cycle-accurate core simulator) separated by memory accesses that are injected into the platform model. ManySim contains a detailed cache hierarchy model, a detailed coherence protocol implementation, an on-die interconnect model and a memory model that simulates the maximum sustainable bandwidth specified in the configuration. All our simulation parameters are shown in Table 2.

CACTI-6.0 [32] is employed for estimating cache area, access latency, and power. We assume a 32nm process for our work. We derive network router power and area overheads from Orion [45]. Each router pipeline is assumed to three stages and link latency is estimated to be three cycles for the grid and tree topologies. We also incorporate a detailed network model into the ManySim infrastructure.

As an evaluation workload, we chose four key multi-threaded commercial server workloads: TPC-C, TPC-E, SPECjbb, and SAP. The bus traces for these workloads were collected on a Xeon MP platform where 8 threads were running simultaneously with the last level cache disabled. To simulate our 16-core system, we duplicate the 8-thread workload to run on 16 cores. This results in true application sharing only between each set of 8 cores. We do offset the address space of each 8-thread trace such that there is no address duplication. Thus, our network latencies for shared pages in the baseline are lower as we do not access the most distant bank, and we will likely see better improvements for a true 16-thread workload. We run these workloads for approximately 145 million memory references. Since we assume an oracle page-coloring mechanism, we annotate our traces off-line with the page color and append the page color bits to the address.

4.2. Baseline Organizations

We consider three different baseline organizations with varying number of dies:

Base-No-PC: A chip with two dies: the bottom die has 16 cores and the second die has 16 1 MB L2 SRAM cache banks organized as S-NUCA but with no explicit page coloring policy. All the banks have a roughly equal probability of servicing a core's request. A grid topology is assumed for the inter-bank network.

Base-2x-No-PC: A chip with three dies: the bottom die has 16 cores and the second and third dies contain SRAM L2 cache banks organized as S-NUCA (no explicit page coloring policy). This organization simply offers twice the cache capacity as the previous baseline at the expense of an additional die.

16 Private MLC Cache banks	Each 128KB 4-way 5-cyc	16 LLC SRAM NUCA Cache Banks	1MB 4-way, 13-cyc
SRAM Active power	0.3W	Page Size	4KB
16 DRAM sector	Each 8MB 32-way 30-cyc	DRAM Active Power	0.42W
Core/Bank Area	$3.5mm^2$	Chip Footprint	$56mm^2$
Process node	32nm	Frequency	4 GHz

Table 2. Simulation Parameters

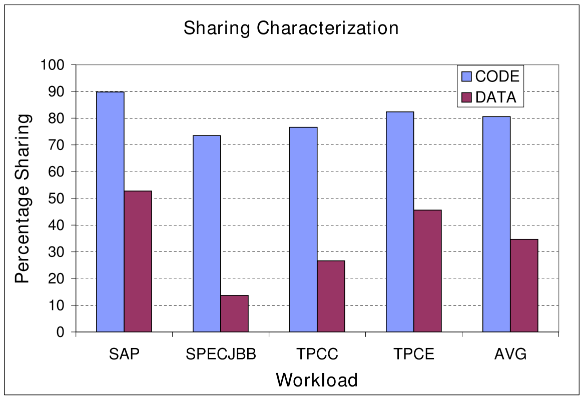


Figure 3. Workload characterization: sharing trend in server workloads

Base-3-level: The DRAM die is used as an L3 UCA cache. The tags for the L3 cache are implemented on the SRAM die, forcing the SRAM L2 cache size to shrink to 0.5 MB. No page coloring is employed for the L2.

4.3. Workload Characterization

We first characterize the server workloads to understand their implications on our proposed techniques. Figure 3 shows the percentage of shared pages in all the workloads. All workloads exhibit high degree of code page sharing. SPECjbb has poor data sharing characteristics and TPC-E and SAP have high data sharing charateristics. We also observed that when threads share code pages, the degree of sharing is usually high. This implies that the "Rp:I + Share4:D" model would have to replicate code pages in most cache banks. However, the working set size of code pages is very small compared to data pages (0.6% on average), but the relative access count of code pages is much higher (57% on average).

4.4. Evaluation of Page Coloring Schemes

In order to isolate the performance impact of page-coloring schemes, we study the effect of these schemes as a function of cache capacity. We use IPC and miss-ratio as the performance metrics. We assume a tree-network topology and vary each cache bank's capacity. All numbers are normalized against the 1 MB baseline bank that employs no page coloring (Base-No-PC), *i.e.*, even private pages may be placed in remote banks.

Figure 4 shows the performance effect of various page-coloring schemes and Figure 5 shows the corresponding miss rates. For cache banks smaller than 2MB, Share4:D+I and Rp:I+Share4:D perform worse than Share16:D+I and no page-coloring case. Since Share4:D+I and Rp:I+Share4:D map all shared pages to the central four banks, these banks suffer from high pressure and high miss rates due to less cache capacity. However, since Share16:D+I distributes shared pages across all banks, it does not suffer from bank pressure. We also observe that Share16:D+I performs slightly better (7%) than the base case (no page coloring) as it is able to optimize access latency for private pages for small cache banks.

For a larger cache bank (8MB), Rp:I+Share4:D performs the best compared to all the other schemes. The overall performance improvement in Share4:D compared to no page-coloring (1MB) baseline is 50%. Rp:I+Share4:D has higher performance as it has lower access latency for all the code pages and does not suffer from high bank pressure in spite of code replication overheads due to available cache capacity. We observe Share4:D+I and Share16:D+I to have comparable performance. We notice that performance of Share16:D+I does not scale with cache capacity as much. Thus, when cache capacity is not a constraint, Rp:I+Share4:D delivers the best performance compared to other schemes and makes an ideal candidate for SRAM-DRAM cache.

With regards to workloads, SPECjbb always performs better than the baseline (with no page-coloring) irrespective of the page coloring scheme employed. Since SPECjbb has low degree of application sharing, it does not suffer from pressure on shared banks and performs better due to communication optimization enabled by page-coloring. However, SAP and TPC-E have poor performance due to high bank pressure for small sized banks as they have high degree of sharing. Figure 5 further affirms our observations and shows miss ratios for various schemes. Clearly, SPECjbb has low miss ratio compared to other workloads. As expected for most cases, Rp:I+Share4:D has higher miss-ratio due to additional conflicts introduced by code replication.

4.5. Reconfigurable SRAM-DRAM Cache

We next evaluate our SRAM-DRAM reconfigurable cache. If an SRAM bank encounters high bank pressure, it enables the DRAM bank directly above. The total available cache capacity in each combined SRAM-DRAM bank is 8.5 MB with 0.5 MB tag space. However, we assume the total per-bank capacity to be 8MB to ease performance modeling. We compare our reconfigurable cache against Base-No-PC and Base-2x-No-PC. Figure 6 shows IPC for various configurations and baselines. When the reconfigu-

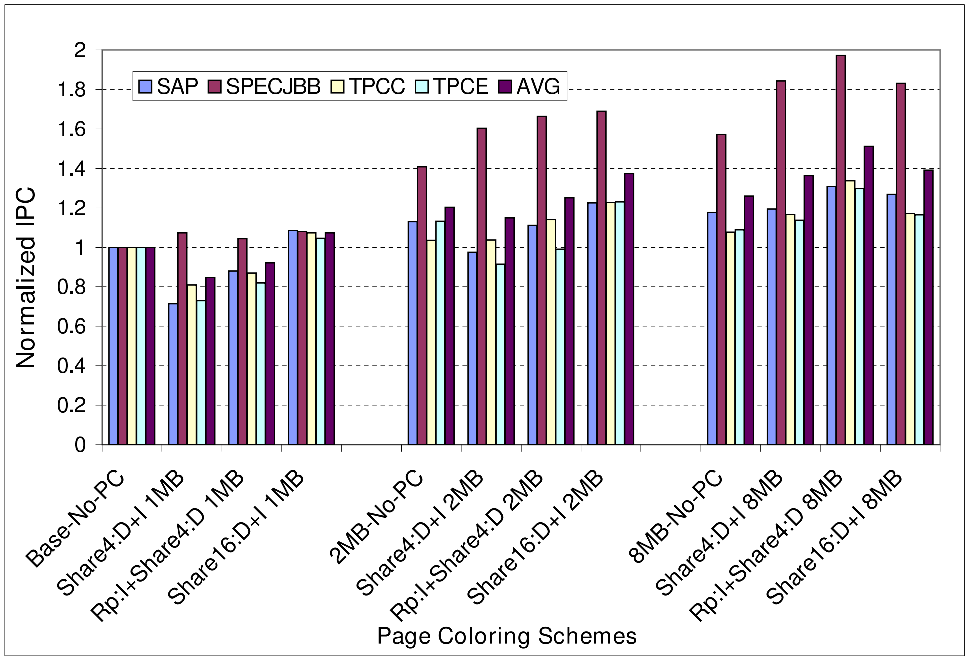


Figure 4. **Performance Evaluation of Page Coloring schemes as a function of Cache Capacity**

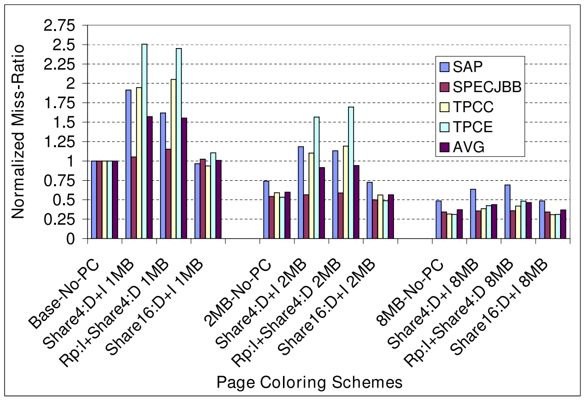


Figure 5. **Miss Ratio of Page Coloring schemes as a function of Cache Capacity**

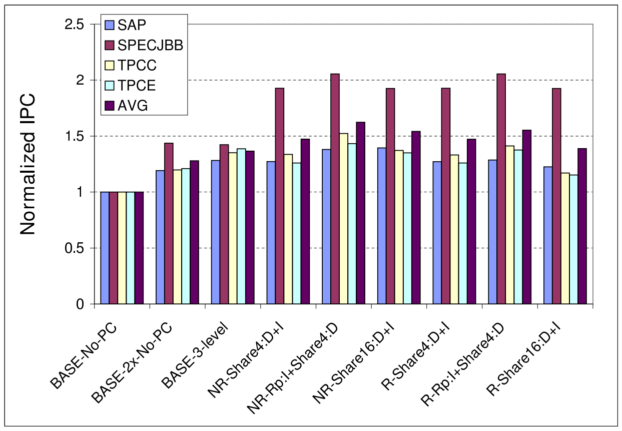


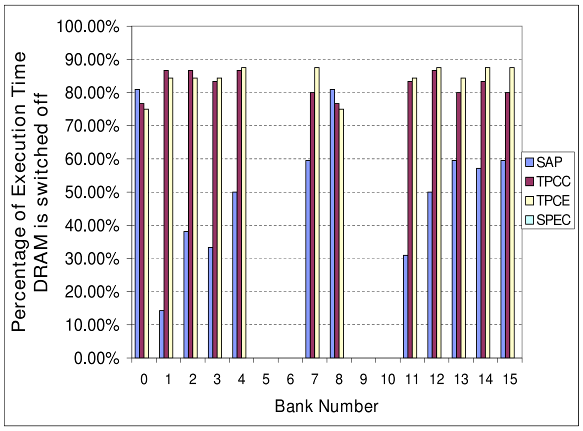
Figure 6. **Performance Evaluation of SRAM-DRAM cache**

ration heuristics are enabled, the model is pre-pended with *R*, and with no dynamic reconfiguration (the DRAM banks are always enabled), the model is pre-pended with *NR*. We assume 100,000 cycles penalty for flushing the caches during reconfiguration.

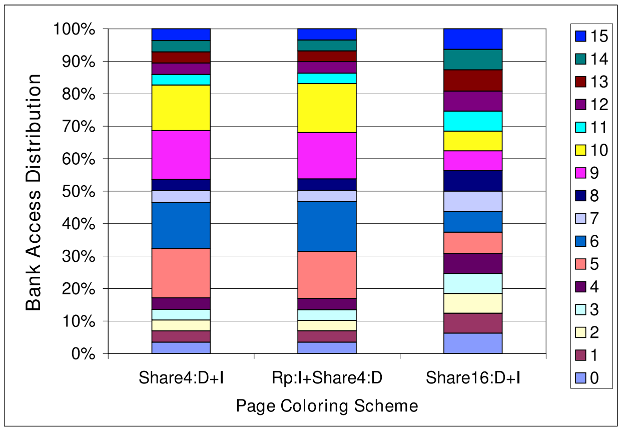
We observe that all workloads on average perform 45-62% better than Base-No-PC when DRAM banks are switched on all the time and 20-35% better compared to Base-2x-No-PC depending upon the page-coloring scheme. When compared to Base-3-level, our NR cases perform 11-26% better depending upon the page coloring scheme. Our best case performance improvement with reconfiguration heuristic enabled when compared to Base-3-level is 19%.

With our reconfiguration heuristic, we observe that DRAM banks are switched on all the time for central shared banks in the Rp:I+Share4:D case. The non-central banks in this model and occasionally in Share4:D+I have DRAM switched off due to low bank pressure. Amongst all the page-coloring schemes, Rp:I+Share4:D yields the best performance with and without reconfiguration heuristic. Since we stall all cores during reconfiguration, the overheads of our heuristic lower the overall performance gain compared to the best case performance with DRAMs enabled all the time. However, we expect this overhead to reduce if the workloads run long enough to amortize the cost of reconfiguration.

Figure 7(a) illustrates the average percentage of time DRAM banks are switched off for each of the cache banks

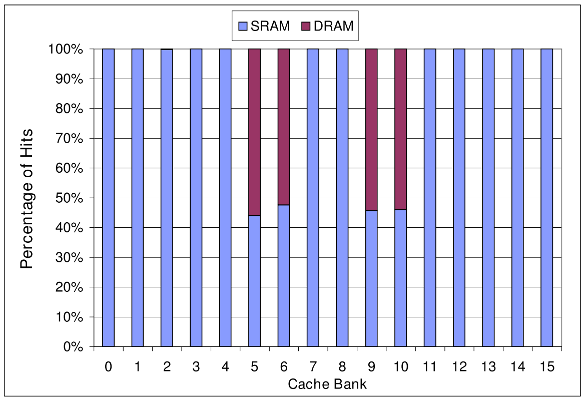


(a) Percentage of total time each DRAM bank is switched off for Rp:I+Share4:D. Banks 5, 6, 9, and 10 are the central banks.



(b) Distribution of accesses based on bank number.

Figure 7. Bank usage.



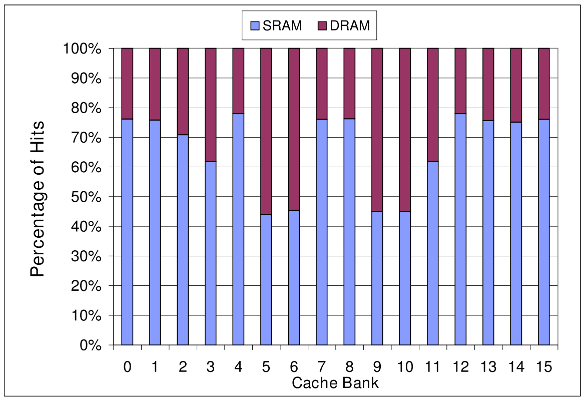


Figure 8. Average percentage of hits in SRAM/DRAM ways for R-Rp:I+Share4:D and NR-Rp:I+Share4:D

for the Rp:I+Share4:D scheme. The DRAM banks are switched off most of the time for non-central banks but shared central banks have DRAM switched on all the time. Amongst various workloads, we found that due to low sharing behavior in SPECJBB, we have high capacity pressure even on non-central banks for this workload. Thus, it has its DRAM turned on most of the time for all page-coloring schemes. SAP workload has multiple program phases leading to maximum number of reconfigurations compared to all the workloads. We show the overall bank access distribution for different page coloring schemes in Figure 7(b). Share4:D+I and Rp:I+Share4:D have maximum accesses to shared banks (5,6,9,10) whereas Share16:D+I has accesses distributed evenly amongst all the banks.

Figures 8(a) and 8(b) show the average distribution of hits in SRAM and DRAM ways for all the cache banks for Rp:I+Share4:D schemes with and without reconfiguration, respectively. Without dynamic reconfiguration, the constant enabling of DRAM banks implies that several L2 look-ups are serviced by the slower and energy-inefficient DRAM banks. Dynamic reconfiguration ensures that DRAM banks are looked up only when bank pressure is high (typically only in central banks). Even the 50% of accesses to DRAM ways in these central banks can perhaps be reduced if we employ some MRU-based heuristics to move blocks from DRAM to SRAM ways.

4.6. Interconnect Evaluation

We find that our page-coloring schemes optimize the network traffic latency significantly enough that the type of network employed does not influence performance much. We expect the performance improvement to be more significant in network topologies with greater number of cores and also where there is true sharing across all nodes. The tree topology performs 2% better than the grid topology in our 16-core simulations with page-coloring enabled. With higher wire and router delays, we expect the performance impact to be more significant. The use of the tree network does lower overall router power overheads due to fewer routers. We observe a 48% reduction in overall network power, compared against a baseline with no page coloring and a grid topology.

The following statistics help explain the power improvements. Figure 9 demonstrates the bank access distribution in a tree/grid network. *Sibling* refers to a bank that is one hop away on the tree topology, *local* refers to the core's

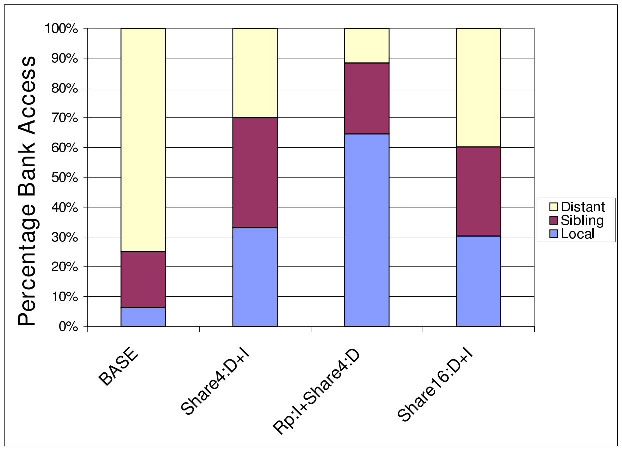


Figure 9. Distribution of bank accesses based on distance

own bank, and *Distant* refers to communication with all other banks. We observe that the baseline with no page coloring sends most of the requests to distant banks and Rp:I+Share4:D maximizes local requests due to code replication.

We also carried out an initial analysis of thermal behavior with Hotspot 4.1 [37] augmented with 3D parameters [26]. Consistent with other papers on 3D stacking of SRAM and DRAM, we observed an increase in temperature of about 7 °C.

5. Conclusions

The central idea in this paper is the design of a cache that enables easy reconfigurability across multiple dies. This central idea is placed in the context of two related essential elements of large cache organization (OS-based page coloring and on-chip network design). OS-based page coloring not only introduces variation in bank working sets, it also enforces a very specific traffic pattern on the network. We take advantage of 3D technology to implement a heterogeneous reconfigurable cache that addresses the problem of variable working sets by selectively providing high capacity at low cost. We advocate a tree topology for the on-chip network because (i) the use of page coloring and S-NUCA avoids search across all banks, (ii) page coloring forces requests to be serviced by local nodes or radiate in/out of central banks, and (iii) reconfiguration prevents a bank from having to spill its data into neighboring banks. The proposed reconfigurable cache model improves performance by up to 19% over the best baseline, along with 48% savings in network power. Our results show that commercial workloads exercise variable pressures on cache banks after page coloring, requiring a dynamic enabling/disabling of DRAM space. The performance of these workloads is very sensitive to cache capacity and quick access to instruction (code) pages. This requires smart page coloring mechanisms as well as the large L2 capacity afforded by dynamic enabling of DRAM banks.

As future work, we plan to improve our reconfiguration heuristics by considering energy/power metrics. We also plan to evaluate other workloads including multi-programmed and parallel emerging workloads. We will also consider ways to maximize SRAM access in our proposed SRAM-DRAM cache.

References

[1] M. Awasthi, K. Sudan, and R. Balasubramonian. Dynamic Hardware-Assisted Software-Controlled Page Placement to Manage Capacity Allocation and Sharing within Large Caches. In *Proceedings of HPCA-15*, Feb 2009.

[2] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures. In *Proceedings of MICRO-33*, pages 245–257, December 2000.

[3] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. A Dynamically Tunable Memory Hierarchy. *IEEE Transactions on Computers*, 52(10):1243–1258, October 2003.

[4] R. Balasubramonian, S. Dwarkadas, and D. Albonesi. Dynamically Managing the Communication-Parallelism Trade-Off in Future Clustered Processors. In *Proceedings of ISCA-30*, pages 275–286, June 2003.

[5] B. Beckmann, M. Marty, and D. Wood. ASR: Adaptive Selective Replication for CMP Caches. In *Proceedings of MICRO-39*, December 2006.

[6] B. Beckmann and D. Wood. TLC: Transmission Line Caches. In *Proceedings of MICRO-36*, December 2003.

[7] B. Beckmann and D. Wood. Managing Wire Delay in Large Chip-Multiprocessor Caches. In *Proceedings of MICRO-37*, December 2004.

[8] J. Chang and G. Sohi. Co-Operative Caching for Chip Multiprocessors. In *Proceedings of ISCA-33*, June 2006.

[9] Z. Chishti, M. Powell, and T. Vijaykumar. Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures. In *Proceedings of MICRO-36*, December 2003.

[10] Z. Chishti, M. Powell, and T. Vijaykumar. Optimizing Replication, Communication, and Capacity Allocation in CMPs. In *Proceedings of ISCA-32*, June 2005.

[11] S. Cho and L. Jin. Managing Distributed, Shared L2 Caches through OS-Level Page Allocation. In *Proceedings of MICRO-39*, December 2006.

[12] W. Dally. Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems (OCIN), 2006. Workshop program and report at `http://www.ece.ucdavis.edu/˜ocin06/`.

[13] A. Dhodapkar and J. E. Smith. Managing Multi-Configurable Hardware via Dynamic Working Set Analysis. In *Proceedings of ISCA-29*, pages 233–244, May 2002.

[14] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and Predicting Program Behavior and its Variability. In *Proceedings of PACT-12*, September 2003.

[15] H. Dybdahl and P. Stenstrom. An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors. In *Proceedings of HPCA-13*, February 2007.

[16] S. Gupta, M. Hilbert, S. Hong, and R. Patti. Techniques for Producing 3D ICs with High-Density Interconnect. In *Proceedings of International VLSI Multilevel Interconnection*, 2004.

[17] Z. Guz, I. Keidar, A. Kolodny, and U. Weiser. Nahalal: Memory Organization for Chip Multiprocessors. *IEEE Computer Architecture Letters*, vol.6(1), May 2007.

[18] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. Keckler. A NUCA Substrate for Flexible CMP Cache Sharing. In *Proceedings of ICS-19*, June 2005.

[19] R. Iyer, L. Zhao, F. Guo, R. Illikkal, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt. QoS Policies and Architecture for Cache/Memory in CMP Platforms. In *Proceedings of SIGMETRICS*, June 2007.

[20] Y. Jin, E. J. Kim, and K. H. Yum. A Domain-Specific On-Chip Network Design for Large Scale Cache Systems. In *Proceedings of HPCA-13*, February 2007.

[21] T. Kgil, S. D'Souza, A. Saidi, N. Binkert, R. Dreslinski, S. Reinhardt, K. Flautner, and T. Mudge. PicoServer: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor. In *Proceedings of ASPLOS-XII*, October 2006.

[22] C. Kim, D. Burger, and S. Keckler. An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches. In *Proceedings of ASPLOS-X*, October 2002.

[23] P. Kundu. On-Die Interconnects for Next Generation CMPs. In *Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems (OCIN)*, December 2006.

[24] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, N. Vijaykrishnan, and M. Kandemir. Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. In *Proceedings of ISCA-33*, June 2006.

[25] C. C. Liu, I. Ganusov, M. Burtscher, and S. Tiwari. Bridging the Processor-Memory Performance Gap with 3D IC Technology. *IEEE Design and Test of Computers*, 22:556–564, November 2005.

[26] G. Loh. 3D-Stacked Memory Architectures for Multi-Core Processors. In *Proceedings of ISCA-35*, June 2008.

[27] G. Loi, B. Agrawal, N. Srivastava, S. Lin, T. Sherwood, and K. Banerjee. A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy. In *Proceedings of DAC-43*, June 2006.

[28] N. Magen, A. Kolodny, U. Weiser, and N. Shamir. Interconnect Power Dissipation in a Microprocessor. In *Proceedings of System Level Interconnect Prediction*, February 2004.

[29] C. McNairy and R. Bhatia. Montecito: A Dual-Core, Dual-Thread Itanium Processor. *IEEE Micro*, 25(2), March/April 2005.

[30] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 Network Architecture. In *IEEE Micro*, volume 22, 2002.

[31] N. Muralimanohar and R. Balasubramonian. Interconnect Design Considerations for Large NUCA Caches. In *Proceedings of the 34th International Symposium on Computer Architecture (ISCA-34)*, June 2007.

[32] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0. In *Proceedings of the 40th International Symposium on Microarchitecture (MICRO-40)*, December 2007.

[33] M. Qureshi and Y. Patt. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *Proceedings of MICRO-39*, December 2006.

[34] P. Ranganathan, S. Adve, and N. Jouppi. Reconfigurable caches and their application to media processing. *Proceedings of ISCA-27*, pages 214–224, June 2000.

[35] Samsung Electronics Corporation. Samsung Electronics Develops World's First Eight-Die Multi-Chip Package for Multimedia Cell Phones, 2005. (Press release from `http://www.samsung.com`).

[36] Semiconductor Industry Association. International Technology Roadmap for Semiconductors 2005. `http://www.itrs.net/Links/2005ITRS/Home2005.htm`.

[37] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, and K. Sankaranarayanan. Temperature-Aware Microarchitecture. In *Proceedings of ISCA-30*, pages 2–13, 2003.

[38] E. Speight, H. Shafi, L. Zhang, and R. Rajamony. Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors. In *Proceedings of ISCA-32*, June 2005.

[39] Tezzaron Semiconductor. `www.tezzaron.com`.

[40] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi. A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies. In *Proceedings of ISCA-35*, June 2008.

[41] M. Tremblay and S. Chaudhry. A Third-Generation 65nm 16-Core 32-Thread Plus 32-Scout-Thread CMT. In *Proceedings of ISSCC*, Febuary 2008.

[42] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In *Proceedings of ISSCC*, February 2007.

[43] K. Varadarajan, S. Nandy, V. Sharda, A. Bharadwaj, R. Iyer, S. Makineni, and D. Newell. Molecular Caches: A Caching Structure for Dynamic Creation of Application-Specific Heterogeneous Cache Regions. In *Proceedings of MICRO-39*, December 2006.

[44] H. S. Wang, L. S. Peh, and S. Malik. A Power Model for Routers: Modeling Alpha 21364 and InfiniBand Routers. In *IEEE Micro, Vol 24, No 1*, January 2003.

[45] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A Power-Performance Simulator for Interconnection Networks. In *Proceedings of MICRO-35*, November 2002.

[46] Y. Xie, G. Loh, B. Black, and K. Bernstein. Design Space Exploration for 3D Architectures. *ACM Journal of Emerging Technologies in Computing Systems*, 2(2):65–103, April 2006.

[47] C. Zhang, F. Vahid, and W. Najjar. A Highly Configurable Cache Architecture for Embedded Systems. In *Proceedings of ISCA-30*, June 2003.

[48] M. Zhang and K. Asanovic. Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors. In *Proceedings of ISCA-32*, June 2005.

[49] L. Zhao, R. Iyer, J. Moses, R. Illikkal, S. Makineni, and D. Newell. Exploring Large-Scale CMP Architectures Using ManySim. *IEEE Micro*, 27(4):21–33, 2007.