3D Tree Cache – A Novel Approach to Non-Uniform Access Latency Cache Architectures for 3D CMPs

Konrad Malkowski, Padma Raghavan, Mahmut Kandemir and Mary Jane Irwin

Department of Computer Science and Engineering
Technical Report
CSE-09-017

December 21, 2009
3D Tree Cache - A Novel Approach to Non-Uniform Access Latency Cache Architectures for 3D CMPs

Konrad Malkowski, Padma Raghavan, Mahmut Kandemir and Mary Jane Irwin
{malkowski, raghavan, kandemir, mji}@cse.psu.edu
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802, USA

Technical Report CSE-09-017

Abstract—We consider a non-uniform access latency cache architecture (NUCA) design for 3D chip multiprocessors (CMPs) where cache structures are divided into small banks interconnected by a network-on-chip (NoC). In earlier NUCA designs, data is placed in banks either statically (S-NUCA) or dynamically (D-NUCA). In both S-NUCA and D-NUCA designs, scaling to hundreds of cores can pose several challenges. In S-NUCA, bank contention can develop when a large number of application threads compete for the same bank. In D-NUCA, with both broadcast and sequential lookup schemes, all banks, all routers, and a significant portion of the NoC links are accessed on a cache miss (on hit sequential scheme may access fewer banks). We propose a new NUCA architecture with an inclusive, tree-based, hierarchical directory (T-NUCA), with the potential to scale to hundreds of cores with performance comparable to D-NUCA at a fraction of the energy cost. We develop two T-NUCA implementations, T-NUCA-2 (binary) and T-NUCA-8 (octal) toward improved performance and energy trade-offs. We simulate these caches on a system with 64-cores, and focus on both single-program and multi-program environments where many applications are competing for cache resources. Our evaluations indicate that in the single-program environment, the T-NUCA-8 cache can reduce execution time by up to 27% over S-NUCA for workloads that have low L2 cache miss rates. In a multi-program environment, where 8 different applications are mapped to 64 cores, T-NUCA-8 can result in significant performance and energy benefits. Relative to S-NUCA, T-NUCA-8 improves the performance and EDP by 70% and 25% respectively, with an energy consumption increase of 150%. Relative to D-NUCA, our T-NUCA-8 reduces network usage by 92%, energy by 87%, and EDP by 87%, at performance cost of 10%. Finally, relative to a 24MB D-NUCA, our 16MB T-NUCA-2 has 7% better performance, and energy and EDP are factors of 8.57 and 9.17 times lower.

I. INTRODUCTION

Increasing process miniaturization, wire delay and power dissipation have brought about a shift from a single super-pipelined, super-scalar processor design to a multi-core processor design where performance is provided by multiple smaller, simpler cores [1], [2]. The increase in the number of cores results in high demand for the limited off-chip memory bandwidth. To offset the impact of these demands, designers are considering CMPs with larger on-chip caches. However, as the cache capacity grows larger, and the transistors become smaller, the access latencies from cores to data locations grow proportionally to their distance due to wire delays [3]–[5].

One approach to reducing the average data access latency is to divide caches into smaller blocks (called banks), each with different access latency, from a given processor core’s perspective. This type of cache architecture is called the non-uniform access cache architecture (NUCA) [5]. The key challenges in the NUCA design are in locating the data and ensuring that the frequently accessed data is always close to the requesting processor, while at the same time facilitating capacity sharing. Various mechanisms such as migration [5], replication [3], [6], cache-cooperation [7], and others [4], [8] are proposed to address these challenges.
The three dimensional (3D) processor design is a complimentary approach to NUCA for lowering wire delays [9]–[11]. In such designs processors are built in layers, either via direct stacking of separate silicon wafers, or through growing respective 3D layers on top of each other. These approaches result in significantly reduced distances between components of a processor, and a corresponding increase in network-on-chip (NoC) bandwidth and throughput, and transistor density.

In this paper, we consider a 3D multi-core system, built using the face-to-back stacking technology [9] where each core is connected to the L2 cache banks via a NoC (see Figure 1 in Section III). In our system, the cores are located on the layer 0, and the L2 cache banks and the NoC are located on layer 1. The key challenges in this architecture are data placement and data lookup. The two best known NUCA implementations are the static NUCA (S-NUCA), and the dynamic NUCA (D-NUCA) [5].

We propose and evaluate a new cache architecture, T-NUCA, with an inclusive, tree-based, hierarchical directory, that attempts to combine the best attributes of D-NUCA and S-NUCA. We develop two T-NUCA implementations, T-NUCA-2 (binary tree directory) and T-NUCA-8 (octal tree directory) toward improved performance and energy trade-offs. In these designs, the cache structure is divided into the hierarchical tag directory and the data storage. The L2 cache data storage cells are laid out in the standard mesh layout on the layer 1. Each bank is associated with a local directory node, and all local directory nodes are located on layer 1. To store the upper levels of our hierarchical directory, we add a service layer. The directory is laid out as a tree on this service layer. At each higher level of the hierarchical directory, the number of directory nodes decreases by the degree of the tree, while the total capacity at each level remains fixed, and each parent directory node contains all the data tags of its children. Finally, there is only one root directory node, which contains tags for all data in the cache.

We evaluate the trade-offs between the 3D S-NUCA, D-NUCA, T-NUCA-2 and T-NUCA-8 designs in two run-time execution scenarios with 64 cores: (i) single-program scenario, where a single benchmark is mapped to all 64 cores, and (ii) multi-program scenario, where 8 different benchmarks are mapped to 8 disjoint sets of 8 cores each. In the single-program scenario our proposed T-NUCA-8 architecture outperforms the S-NUCA cache for workloads with low L2 cache misses by an average of up to 27%. For codes with medium miss rates in L2 cache, the T-NUCA-8 performs comparably to the S-NUCA cache. On average, for T-NUCA-8, the performance is 2% lower than for S-NUCA over all benchmarks. Relative to D-NUCA, the T-NUCA-8 architecture reduces energy consumption by an average of 77% and EDP by 75%, while offering comparable performance on all benchmarks.

In the multi-program scenario, relative to D-NUCA, our T-NUCA-8 design reduces the energy consumption by 87% and EDP by 87% at a performance degradation cost of 10%. Relative to S-NUCA, our T-NUCA-8 design reduces the execution time by 70% and EDP by 25%, at a cost of increased energy consumption of 150%. The performance benefits of T-NUCA-8 come at the expense of extra transistors for its implementation. These overheads are analyzed in Section III-A. To evaluate the potential of using the transistor overhead to improve D-NUCA and S-NUCA, in Section VI we consider multi-program environment performance on D-NUCA and S-NUCA, with a commensurably larger capacity or a larger NoC bandwidth. Our results indicate significant energy increases for D-NUCA and S-NUCA, with minimal or no performance gains in both cases. For example, relative to a 24MB D-NUCA, our T-NUCA-2 has 7% better performance, at a factor of 8.57 lower energy cost.

The remainder of this paper is organized as follows. In Section II, we describe S-NUCA and D-NUCA cache configurations used as reference points in evaluating the performance and energy efficiency of our T-NUCA-2 and T-NUCA-8 architectures. In Section III, we provide details of our T-NUCA-2 and T-NUCA-8 designs, including an analysis of overheads incurred by the directory structure. Section IV presents our experimental setup and tools used to evaluate S-NUCA, D-NUCA, T-NUCA-2 and T-NUCA-8, and Sections V and VI present and discuss our results. Section VII discusses related work on the NUCA design. Finally, in Section VIII, we summarize our results and provide concluding remarks.
II. THE REFERENCE S-NUCA AND D-NUCA CONFIGURATIONS

In this section, we discuss our reference NUCA configurations: S-NUCA, and D-NUCA with broadcast and data migration. These two architectures represent the opposing ends of the NUCA design spectrum. S-NUCA is a conservative design which reduces the energy consumption and the network use at the cost of performance, while D-NUCA is an aggressive design which trades energy consumption and network use for high performance. Our S-NUCA and D-NUCA cache designs are based on Kim et al.’s work [5].

S-NUCA uses a static data placement and lookup method, where banks are chosen based on a subset of bits in the data address. Due to static data placement in the cache, the S-NUCA design is energy efficient, and has a low network link usage. However, it does not guarantee high performance due to lack of migration (data can be far away from the requesting core, many cores can be competing for the same bank).

In our D-NUCA design, we focus on achieving maximum performance. Thus, our L2 cache consists of one bank-way shared among all the cores with dynamic mapping of data to banks. In particular, on a cache miss, a bank is randomly selected to hold the data. On each access to the L2 cache, a broadcast is issued to locate the bank in which the data is located. The miss is determined after a time-out equal to twice the longest access latency. During the broadcast all banks are activated and checked for the requested data. On each read hit, the data is migrated one bank closer to the core that requested it. This migration is performed in the X direction first and then in the Y direction. During migration contents of target and source cache lines are exchanged among banks. On a miss replacement dirty data is written back to main memory, and clean data is overwritten. Finally, for both S-NUCA and D-NUCA we employ the MESI protocol [12] for L1 and L2 cache coherency. All decisions regarding the MESI protocol are handled at the level of the L2 bank containing the data.

III. A HIERARCHICAL, TREE-BASED DIRECTORY NUCA ARCHITECTURE

In this section, we present our T-NUCA-2 and T-NUCA-8 architectures. Figure 1 depicts our target NUCA configuration on a 3D CMP system with 64 cores. In this design the processor consists of two layers, the compute layer (layer 0), which holds 64 compute cores, and the cache layer (layer 1), which holds the 64 shared L2 cache banks, 64 routers and NoC. Each core’s L1 cache is directly connected to one router via a 3D via. The only way to access the data located in the L2 banks is via the mesh-based NoC interconnect. The challenge in this design is in determining where to place the data in cache and how to locate the data on an L1 cache miss. We propose a novel approach to solving the data lookup problem in a D-NUCA. Instead of relying on the NoC to locate the data by broadcasting requests to all banks, we add an inclusive, tree-based, hierarchical directory for data lookups and updates, on an additional silicon layer (service layer) above the cache layer (layer 1 in Figure 1). The local L2 bank directories are located on the same layer as the data arrays. The higher levels of the directory are located on the service layer. The local directories are connected to the upper level by 128 bit 3D vias (two 64 bit channels, one in each direction) and links. In our design, we separate the data lookup stage of the cache access from the data access stage. As a result, when we search for the required data, only directory tags are accessed. Consequently, in our design, data accesses and tag lookups do not contend for the same network resources, thereby helping on-chip parallelism as well. The tree can be of arbitrary degree. In this paper we consider a binary tree based directory (T-NUCA-2) and an octal tree based directory (T-NUCA-8). The conceptual design of our T-NUCA-2 is shown in Figure 2.

Our T-NUCA designs are motivated by two factors. (i) Analysis of the data access distance for our D-NUCA design with random placement on miss and migration. We observe that in a multi-program environment (discussed in detail in Section IV), 94% of data requests are satisfied within the distance of 4 hops from the requesting core. In the single-program environment, for example for EQUAKE, close to 63% of requests are satisfied...
with 4 hops. Thus, despite the significant locality of accesses, the D-NUCA floods the network with data requests, and activates all banks, all routers, and majority of links in the cache (See Figure 3).

(ii) Observation that programs written for multi-core processors will have to be designed to conserve the
main memory bandwidth, i.e., have working sets that fit in lowest level of cache, and low degree of misses. For such codes S-NUCA design could result in significant overheads due to conflict misses, while D-NUCA will result in significant energy overheads due to bank activations on each access.

A. T-NUCA-2 and T-NUCA-8 Design

We begin with a detailed discussion of the T-NUCA-2 design, and then we discuss the differences between the T-NUCA-8 and T-NUCA-2 designs. In our T-NUCA-2 design, the data arrays are arranged as a 2-D mesh and connected via a NoC. Each data array is associated with one local directory node located at the lowest level of the tree (level 0). At each successive level up in the tree, the number of separate directory nodes is cut in half, and their capacity is doubled. As our hierarchical directory is inclusive, all addresses at the lower levels of the tree are replicated at the higher levels of the tree. We double the capacity of higher level directory nodes by doubling their associativity.

Due to tag replication in the tree directory, our tree based T-NUCA-2 design has a transistor overhead associated with it. Consider an L2 cache with \(N\) banks in a \(\sqrt{N}\) by \(\sqrt{N}\) grid, where each bank of the cache has \(C\) bits dedicated to address tag storage. Our hierarchical directory is built as a binary tree and thus requires \(\log_2 N\) levels (\(\log_2 \sqrt{N}\) in each grid direction). Each level of the tree requires \(C \times N\) bits to store all address tags present in cache. Thus, the bit cost of our hierarchical directory is \(C \times N \times (\log_2 N + 1)\), compared to a cost of \(C \times N\) for storing all address tags in an S-NUCA or D-NUCA. The cost of storing the data arrays in all three architectures is \(A \times N\), where \(A\) is the number of bits dedicated to storing all cache lines. Both \(A\) and \(C\) are functions of \(N\), and we expect them to grow at least linearly with the number of cores on chip.

The bit overhead associated with the hierarchical directory in T-NUCA is expressed by

\[
\Phi = \frac{C \times N \times (\log_2 N + 1) + A \times N - C \times N - A \times N}{C \times N},
\]

which simplifies to \(\Phi = \frac{C \times A}{C \times \log_2 N - A}\). It can be approximated by \(\Phi = \frac{C \times \log_2 N + A}{A}\). More specifically, consider a 16MB cache with 64 banks, 64 byte wide cache lines, 4 way set-associativity and 48-bit addresses. In such a design, each cache set consists of 2,048 bits for data storage, and 32 bits for each address tag (128 bits total). Thus, a 16MB D-NUCA (S-NUCA) requires 134 million bits for data and 8 million bits for address tags. Our T-NUCA-2 design requires the same number of bits for data storage as D-NUCA, and 59 million for the directory (due to directory overheads). The total bit overhead of our T-NUCA is 35% over D-NUCA with comparable cache data capacity. As the cache capacity grows to accommodate more cores, we observe that the tree
directory overheads will slowly increase with each doubling of the number of cache banks. We expect the tree directory to grow in height by one level with each doubling of number of banks, thus growing at the rate of \(\log_2 N\), where \(N\) is the number of banks.

The growth of bit overhead of T-NUCA-2 can be reduced by changing the design of the hierarchical tree directory. Thus we consider a T-NUCA-8 architecture which is based on the octal tree hierarchy and for 64 banks consists of only 3 levels. In the T-NUCA-8 design, the root directory is connected to 8 intermediate directories, and each intermediate directory is connected to 8 local directories. The bit overheads for the T-NUCA-8 architecture are 11% over D-NUCA and S-NUCA.

Due to sparsity of the T-NUCA-2 and T-NUCA-8 tree directory structures, we estimate that approximately 65% and 89% respectively, of the service layer would be unused by our designs. However, the unused area would decrease as the number of banks, and the capacity of the cache increase. In addition, the unused space could be used for other purposes, such as, thermal sensor, debugging interfaces [11], power control circuitry, or thermal vias (i.e., vias that are made of copper, are not connected to anything, and serve to help evacuate heat out of the processor) [13]. Alternatively, the unused space could be used for improving processor reliability, by duplicating some of the cores [14] or their crucial components. The low overheads of T-NUCA-8 also suggest that this design can be adapted to standard 2D CMP design with minimal grid layout deformation.

**B. T-NUCA-2 and T-NUCA-8 Access**

In this section, we discuss in detail how data accesses in T-NUCA-2 are handled. The accesses in T-NUCA-8 follow the same pattern. On an L1 cache miss, the directory node associated with local L2 bank is checked first. On a miss there, the node’s parent is queried, and so forth, until either the requested address is found in cache, or the root directory node is reached. If the data is located, the request is sent via the directory tree to the data’s owner bank. The owner bank then sends the data to the requesting core over the mesh NoC. The target and source cache lines are exchanged between banks and the information in the hierarchical directory is updated. If the data is not found at the root directory node, a least recently used cache line in a randomly selected bank is chosen as a victim and evicted.

Our design performs the hierarchical directory status updates in the background (without involving the mesh NoC). The only exception to this rule are MESI coherence operations, which stall the cache request until the coherence protocol requirements are met. In addition, all invalidations, write-backs and MESI status updates in L1 caches take place over the mesh network. Lastly, writes from L1 to L2 require an ACK message to be sent to the writer upon completion of the write, for internal state maintenance. With use of buffers the writer does not have to wait for the write to complete.

**IV. EXPERIMENTAL SETUP**

In this paper, we use two Serengenti machine configurations in SIMICS full system simulator [15] to compare T-NUCA-2 and T-NUCA-8 against S-NUCA and D-NUCA. (i) We use an 8 processor configuration to obtain memory access traces for the SPEC OMP [16] benchmarks for multi-program environment evaluation. The SPEC OMP benchmarks (ammp, art, equake, fma3d, gafort, galgel, mgrid and swim) are fast forwarded 4 billion instructions (on each core). Each benchmark is executed for 250 million instructions per core, with the reference problem size. The SIMICS memory traces are executed by our S-NUCA, D-NUCA, T-NUCA-2 and T-NUCA-8 cache simulators. (ii) We use a 64 processor configuration to directly evaluate single-program performance of the SPEC OMP and PARSEC [17] benchmarks with S-NUCA, D-NUCA, T-NUCA-2 and T-NUCA-8 caches in SIMICS. The SPEC OMP benchmarks are fast forwarded 4 billions instructions, and PARSEC benchmarks are fast forwarded to their region of interest. Each benchmark is then executed for 4 billion instructions.

All simulators implement a 64 core processor architecture, where each core is directly connected via a router to a local bank of the L2 cache. For all configurations we consider a 16MB L2 cache,
divided into 64 256KB banks. The banks are four-way associative, with a 64 byte cache line. There are 1024 cache sets in each bank. Each core contains a 32KB L1 cache, which is 4 way associative and has access latency of 1 cycles. The main memory access latency is 200 cycles. In our simulation environment we idealize the hardware design and assume no network contention.

We use CACTI5.3 [18] to obtain the cache access latency and energy estimates for a 256KB bank in a 45nm technology. We simulate the D-NUCA and S-NUCA bank designs assuming 1 read/write port design. In the T-NUCA-2 and T-NUCA-8 designs we calculate the access latencies and energy consumption values of the hierarchical directory, data array access, and mesh interconnect separately. We use latency and energy values for 1 read/write port design for data arrays (as data can be transferred through the NoC mesh). We use latency and energy values for the tag array portion of 1 read/write port, 1 read port, 1 write port, 256KB bank design for local tag directory. The extra read and write ports are used for connecting to the upper levels of tree directory. We use latency and energy values for tag array of a 3 read ports and 3 write ports for T-NUCA-2 (a pair of ports for each link in the tree node), and 9 read ports and 9 write ports for T-NUCA-8, 256KB bank design to simulate the tree tag directories.

The energy consumption of each directory at each successive higher level in the tree is increased, to account for the increase of the directory capacity. We also scale the tree link delays accordingly. Our tree is laid out in such a way that directories in the X direction are connected first and then the directories in the Y direction. The resulting link latencies for T-NUCA-2 as we go from level $i$ to level $i+1$ are 1, 1, 2, 1, 1, 2 cycles, respectively. For T-NUCA-8 the latency to go from level 0 to level 1 of the directory are 4, 3, 2, 1, 2, 3, 4 cycles starting at the leftmost local directory in each row. The latencies to go from level 1 to root directory are 4, 3, 2, 1, 1, 2, 3, 4 cycles starting at the topmost level 1 directory in the column. At each level of the tree directory we use cacti5.3 derived access latencies for each directory. For T-NUCA-2 the access latencies are 2, 3, 3, 4, 4, 6, and 8 cycles, starting with the level 0 local directory. For T-NUCA-8 the directory access latencies are 2, 4, and 12 cycles, starting with the level 0 local directory.

We use CACTI6.0 [19] to obtain latency and energy values for the NoC routers and links. We modify the CACTI model to simulate a 6-port router. We account for latency of traversing the via’s in the chip by rounding up the link latency between local directory and the first directory on the service layer to one cycle. Likewise we assume that the power for traversing the via is included in the link power estimate from cacti (the length of a via is 5 to 50 µm [10], while a length of a link in our cache is between .75 and 1.5mm, making the contribution of the via’s to the total link power and latency negligible). We simulate the tree links using two 64-bit unidirectional busses, one to send requests and information up the tree, and the other to send information down the tree. The values of all cache parameters are listed in Table I.

We consider two execution scenarios in our experiments, (i) single-program and (ii) multi-program. In the first scenario, we simulate a single program with an exclusive access to all 64 cores. In the second scenario, we simulate eight programs running concurrently on the 64 cores, with each application assigned to a contiguous block of 8 cores. Within this scenario, we consider the impacts of thread-to-core mapping on the T-NUCA by evaluating three such mappings: (i) the row-wise mapping (our base configuration), (ii) the column-wise mapping, and (iii) the blocked mapping (where data is assigned to a rectangular block of cores and the block aspect ratio is kept as close to 1 as possible, i.e., a square). We also consider opportunities for improving S-NUCA and D-NUCA performance relative to T-NUCA-2 by dedicating the T-NUCA-2 transistor overhead to higher S-NUCA and D-NUCA NoC bandwidth or cache capacity.

V. RESULTS FOR THE SINGLE-PROGRAM EXECUTION SCENARIO

We evaluate the T-NUCA-2 and T-NUCA-8 designs in a single-program scenario first. For our experiments we use a subset of SPEC OMP (ammp, art, applu, equake, gafort, galgel, mgrid) and PARSEC (blackscholes, facesim, freqmine, fluidanimate, swaptions, streamcluster) benchmarks. We begin by investigating the average number of links, and the
average number of hops per L2 access used by S-NUCA, D-NUCA, T-NUCA-2 and T-NUCA-8 in the single-program execution scenario in Figure 4 (see left plot). When compared against S-NUCA, the T-NUCA-2 and T-NUCA-8 lower the average hop count by 31% on average. However, due to directory overheads, the average number of links used per L2 access increases on average by 94% for T-NUCA-2 and on average by 1% for T-NUCA-8. With respect to D-NUCA, T-NUCA-2 reduces the average number of links used by 59% on average, and T-NUCA-8 reduces the average number of links used by 79% on average, while maintaining the same average number of hops per L2 access. In Figure 5, we compare the energy, EDP and execution time and miss rate results for S-NUCA, D-NUCA, T-NUCA-2 and T-NUCA-8. We observe that for codes that exhibit good L2 cache locality and low miss rates both T-NUCA-2 and T-NUCA-8, as well as D-NUCA outperform the S-NUCA...
Fig. 5. Single-program scenario - Time (Top Left), miss rate (Top Right), energy (Bottom Left) and EDP (Bottom Right) results for SPEC OMP and PARSEC benchmarks on S-NUCA, D-NUCA, T-NUCA-2 and T-NUCA-8. Results normalized with respect to S-NUCA.

cache architecture. For example, T-NUCA-2 and T-NUCA-8 improve the performance of 5 of 13 benchmarks over the S-NUCA. For example, for gafort, our T-NUCA-2 outperforms the S-NUCA by 25%, and T-NUCA-8 by 26%. For codes with moderate miss rates, all of the dynamic designs perform comparably or worse than the S-NUCA due to design overheads (directory traversal in T-NUCA-2 and T-NUCA-8 and miss time-out in case of D-NUCA). On average across all benchmarks T-NUCA-2 suffers 9% performance degradation over S-NUCA and T-NUCA-8 suffers 3% performance degradation.

We observe that relative to S-NUCA the D-NUCA energy costs are a factor of 10 to 35 times higher, relative to T-NUCA-2 a factor of 3.3 (on
average) times higher (Figure 5), and relative T-
NUCA-8 a factor of 4.3 (on average) times higher.
The EDP costs of D-NUCA are a factor of 25 (on
average) higher than S-NUCA, a factor of 2.9 (on
average) higher than T-NUCA-2, and a factor of 4
(on average) higher than T-NUCA-8. These over-
heads are due to network and bank active energy
consumed by D-NUCA on each broadcast.

The energy costs of T-NUCA-2 and T-NUCA-8
are a factor of 7.6 and 5.9 (on average) higher than
the energy costs for S-NUCA. Similarly, EDP costs
for T-NUCA-2 and T-NUCA-8 are on average a
factor of 8.8 and 6.4 times higher than for S-NUCA.
These increases are caused by the active energy
spent in our directory structure for tag updates
and tag lookups (The plot in Figure 6 shows the
distribution of levels where data read requests were
satisfied for all 64 cores). We believe that the energy
consumption of T-NUCA could be further reduced
by combining some of the directory updates.

VI. RESULTS FOR THE MULTI-PROGRAM
EXECUTION SCENARIO

We now compare the S-NUCA, D-NUCA, T-
NUCA-2 and T-NUCA-8 designs in the multi-
program execution scenario. In this scenario, our
system is executing 8 applications from the SPEC
OMP benchmark suite on 64 cores, and each appli-
cation is assigned 8 cores in a row of the chip. We
begin by considering link use and average hop count
in Figure 7 (left plot). We observe that relative to
D-NUCA, our T-NUCA-2 reduces the network link
use by 81%, and T-NUCA-8 reduces network link
use by 92%, while maintaining the same average
number of hops that data needs to travel on an
L2 cache read. Additionally, when compared to S-
NUCA, T-NUCA-2 and T-NUCA-8 use 56% and
82% respectively less links on average and have a
lower average hop count due to data migration. We
next consider mesh NoC and hierarchical directory
activity in T-NUCA-2 and T-NUCA-8. We observe
that T-NUCA-2 and T-NUCA-8 reduce the network
activity and energy consumption of the system by
taking advantage of data locality. As shown in the
middle plot of Figure 7, an overwhelming majority
of all read accesses in the L2 cache is resolved at the
local directory level (level 0), and the average direc-
tory level where data is found is 0.52 for T-NUCA-
2 and 0.25 for T-NUCA-8. When we analyze the
performance of the busiest core in the simulation
(the one that executed the longest), we see that the
majority of L2 reads is resolved at the root directory
level. This behavior indicates that on average the
data needed by the busiest core is located far from
it. This is validated by the average hop count for
the busiest core of 5.75 hops, while on average for
the whole CMP the average hop count is just 0.63
hops for both T-NUCA-2 and T-NUCA-8.

In Figures 7 and 8, we consider impacts on
performance, energy, EDP and L2 miss rates. Our
T-NUCA-2 design improves the performance by
nearly 68% when compared to S-NUCA cache,
and the T-NUCA-8 design improves performance
by 70% relative to S-NUCA. Relative to D-NUCA,
both T-NUCA-2 and T-NUCA-8 degrade perfor-
mance by 16% and 10% respectively, at energy
savings of 83% and 87%, and EDP reduction of
79% and 87% respectively. Relative to S-NUCA,
T-NUCA-8 increases the energy consumption by
150% (due to directory overheads) and reduces
the EDP by 25%. T-NUCA-2 increases the energy
consumption by 250%, and EDP by 13%.

A. Sensitivity to Thread-to-Core Mapping

In our evaluation thus far, we have used a row-
wise thread-to-core mapping, which might unduly
benefit our T-NUCA (tree is built row-wise first).
We now consider T-NUCA with a column-wise and
a blocked thread-to-core mapping. In the column-
wise mapping, 8 cores in a column are assigned to
each application. In the blocked mapping, 8 cores in
a 4 by 2 rectangle are mapped to each application.

We observe that for all of our designs, the best
performance is achieved for the blocked mapping
(figures omitted for space). For T-NUCA-2 with
blocked mapping, the execution time is reduced by
6%, relative to T-NUCA-2 with row-wise mapping,
energy is increased by 2% and EDP is reduced by
5%. For T-NUCA-8 with blocked mapping, the exe-
cution time is reduced by 8%, relative to T-NUCA-8
with row-wise mapping, energy is decreased by 1%
and EDP is reduced by 8%. These improvements
are due to a higher fraction of data requests that
are satisfied at the lower levels of the directory tree
(figure omitted for brevity).
B. Transistor Budget Trade-offs

In this section, we consider a scenario where the T-NUCA-2 directory transistor overheads are used to increase the NoC link width of S-NUCA and D-NUCA from default 64-bits, to 128-bits and 256-bits, or to increase capacity from 16MB to 24MB, under the row-wise thread-to-core mapping scheme. Our results indicate that increasing the bandwidth of the mesh does not result in significant performance improvements for S-NUCA or D-NUCA (approximately 3%). Furthermore, increasing the link width of the NoC results in significantly increased energy consumption and higher EDP values, making increasing the bandwidth of S-NUCA and D-NUCA less attractive compared to our T-NUCA-2 and T-NUCA-8 designs. For example, a 64 bank 24MB D-NUCA has 7% lower performance than our 16MB T-NUCA-2 cache, at a factor of 8.57 higher energy consumption, and a factor of 9.17 higher EDP. The higher energy costs of 24MB D-NUCA and S-NUCA, and their lower performance are the result of longer bank access times (7 cycles instead of 5), and higher energy costs for accessing a bank.

VII. RELATED WORK

In this section, we discuss earlier work on the NUCA architectures. The original NUCA cache architecture together with several architectural variants was proposed by Kim et al. [5] as a way of handling increasing wire latencies in large cache structures by dividing the cache structure into banks, and allowing the access latency to be dependent on the distance between the bank and the processor. Huh et al. [8] further extended the work on NUCA to accommodate the case of multiple cores sharing the overall NUCA cache. Chisthi et al. [20] proposed an alternative design called NuRAPID, in which the cache is divided into few large structures instead of many small structures for higher reliability and
Fig. 7. Multi-program scenario. Left: Average distance in hops of where data is found on a L2 read (top subplot), and the average number of links utilized per access (bottom subplot). Middle: Distribution of accesses satisfied at each level of the tree directory for the whole CMP for T-NUCA-2 (top subplot) and T-NUCA-8 (bottom subplot). Right: Distribution of accesses satisfied at each level of the tree directory for the busiest core for T-NUCA-2 (top subplot) and T-NUCA-8 (bottom subplot).


efficiency, and lower data migration rates. Christi et al. [4], further extended their work to accommodate a limited number of cores.

Chang et al. [7] introduced the concept of cooperative caching in multi-core processor systems, where each core has a local L2 cache, and coherency and sharing are achieved by all processors listening in on all the L2 traffic and cooperating to reduce the conflicts and increase the overall capacity. Liu et al. [21] introduced a fully shared L2 NUCA cache where the most frequently shared data elements are placed in a central bank located centrally to all cores. Guz et al. [22] on the other hand proposed the Nahalal architecture in which the L2 cache is divided into two distinct regions: (i) private L2 region for each core and (ii) shared L2 region among all cores. In this design, the shared banks are in the center of the processor, the cores enclose the shared elements, and private L2 caches are located in the periphery on the processor. While these approaches are very effective for a small number of processors, none of them addresses the problem of cache sharing and cache access in architectures where we have 64 or more cores.
A complimentary approach to NUCA for reducing impact of long wires in small feature size technology involves migration to a 3D processor manufacturing process and is a subject of extensive research. Loh et al. [9] presents a good overview of the field, and discusses the advantages and disadvantages of shifting to a 3D processor design. Vaidyanathan et al. [23] show the potential performance and power benefits of 3D technology under thermal constraints. Li et al. [10] discuss in detail the design of 3D processors and various 3D network implementations, and propose the use of communication pillars based on dTDMA busses, across the wafer layers for interlayer communication.

Trees have been used successfully for a long time in scientific and high-performance computer designs. For example, many of the high speed networks for large scale systems are based on fat trees [24], and many of the global data operations map very nicely to a tree network, e.g., broadcast, all-to-all, scatter/gather operations in MPI [25]. Tree structures have also been used successfully in shared distributed memory systems [26] for storing the page sharing information where the number of nodes is large and a full map sharing directory would be prohibitively expensive. In a more recent work, Eisley et al. [27] proposed in network virtual trees for maintaining sharing and coherency information. Finally, in many array/loop intensive applications, trees form the basis of computation. They convey information about data and computational dependencies, as well as allow for significant performance optimizations [28]. All of these approaches motivate us for exploring the T-NUCA cache architecture in the context of CMPs.

VIII. CONCLUSIONS

We propose a new cache architecture, in which the data and address tag bits are separated, and the address tags are stored in an inclusive, hierarchical, tree-based directory (T-NUCA-2 and T-NUCA-8). At each node in our hierarchical directory the entries of the children nodes are replicated. Consequently, the root node contains all address tags and has a global view of the cache contents. We evaluate the performance and energy efficiency of T-NUCA-2 and T-NUCA-8 against S-NUCA and D-NUCA on 64 cores in two execution scenarios, (i) a single-program scenario, where one application is assigned all cores on the chip, and (ii) a multi-program scenario, where eight applications run concurrently in disjoint core partitions. In the multi-program scenario, relative to D-NUCA, our T-NUCA-8 design reduces energy by 87% and EDP by 87% at a cost of 10% performance degradation. Relative to S-NUCA, our T-NUCA-8 design reduces the execution time by 70%, and EDP by 25%, at the cost of increased energy consumption of 150%. These results indicate that T-NUCA-8 presents an improved trade-off between performance and energy on parallel workloads relative to S-NUCA and D-NUCA. Further improvements in the energy efficiency of T-NUCA-8 will likely be possible by combining some of the tree updates.

REFERENCES


“Cacti 5.3,” http://www.hpl.hp.com/research/cacti/?jumpid=reg_R1002_USEN.


