Spatial Locality Speculation to Reduce Energy in Chip-Multiprocessor Networks-on-Chip

Hyungjun Kim*, Boris Grot†, Paul V. Gratz* and Daniel A. Jiménez‡
*Department of Electrical and Computer Engineering, Texas A&M University
†Institute of Computing and Multimedia Systems, EPFL
‡Department of Computer Science, The University of Texas at San Antonio
*{hyungjun,pgratz}@tamu.edu †boris.grot@epfl.ch ‡dj@cs.utsa.edu

Abstract—As processor chips become increasingly parallel, an efficient communication substrate is critical for meeting performance and energy targets. In this work, we target the root cause of network energy consumption through techniques that reduce link and router-level *switching activity*. We specifically focus on memory subsystem traffic, as it comprises the bulk of NoC load in a CMP. By transmitting only the flits that contain words predicted useful using a novel spatial locality predictor, our scheme seeks to reduce network activity. We aim to further lower NoC energy through microarchitectural mechanisms that inhibit datapath switching activity for unused words in individual flits. Using simulation-based performance studies and detailed energy models based on synthesized router designs and different link wire types, we show that (a) the prediction mechanism achieves very high accuracy, with an average rate of false-unused prediction of just 2.5%; (b) the combined NoC energy savings enabled by the predictor and microarchitectural support are 36% on average and up to 57% in the best case; and (c) there is no system performance penalty as a result of this technique.

1 Introduction

While process technology scaling continues providing more transistors, the transistor performance and power gains that accompany process scaling have largely ceased [1]. Chip-multiprocessor (CMP) designs achieve greater efficiency than traditional uniprocessors through concurrent parallel execution of multiple programs or threads. As the core count in chip-multiprocessor (CMP) systems increases, networks-on-chip (NoCs) present a scalable alternative to traditional, bus-based designs for interconnection between processor cores [2]. As in most current VLSI designs, power efficiency has also become a first-order constraint in NoC design. The energy consumed by the NoC itself is 28% of the per-tile power in the Intel Teraflop chip [3] and 36% of the total chip power in MIT RAW chip [4]. In this paper we present a novel technique to reduce energy consumption for CMP core interconnect leveraging spatial locality speculation to identify unused cache block words. In particular, we propose to predict which words in each cache block fetch will be used and leverage that prediction to reduce dynamic energy consumption in the NoC channels and routers through diminished switching activity.

1.1 Motivation

Current CMPs employ cache hierarchies of multiple levels prior to main memory [5], [6]. Caches organize data into *blocks* containing multiple contiguous words in an effort to capture spatial locality and reduce the likelihood of subsequent misses. Unfortunately, applications often do not fully utilize all the words fetched for a given cache block, as recently noted by Pujara et al. [7].

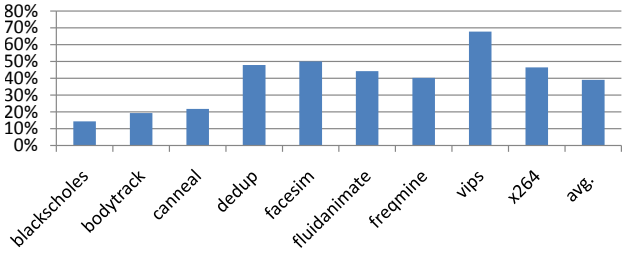


Fig. 1. Percentage of 64-byte block, cache words utilized per block in the PARSEC multithreaded benchmarks.

Figure 1 shows the percentage of words utilized in the PARSEC multithreaded benchmark suite [8]. On average, 61% of cache block words in the PARSEC suite benchmarks will never be referenced and represent energy wasted in transference through the memory hierarchy [1]. In this work, we focus on the waste associated with traditional approaches to spatial locality, in particular the wasted energy and power caused by large cache blocks containing unused data.

1.2 CMP Interconnect

Networks-on-chip (NoCs) purport to be a scalable interconnect to meet the increasing bandwidth demands of future CMPs [2]. NoCs must be carefully designed to meet many constraints. Energy efficiency, in particular, is a challenge in future NoCs as the energy consumed by the NoC itself is a significant fraction of the total chip power [3], [4]. The NoC packet datapath, consisting of the link, crossbar and FIFOs, consumes a significant portion of interconnect power, 55% of network power in the Intel Teraflop chip [3].

Existing NoCs implement channels with relatively large link bit-widths (>=128 bits) [9], [10], a trend expected to continue as more wire density becomes

1. Versus 64-byte lines, 32-byte lines reduces unused words to 45%, however it increases AMAT 11% (See Section 5.3).

available in future process technologies. These high-bandwidth link wires reduce the latency of cache block transmission by allowing more words to be transferred in each cycle, minimizing serialization latency for large packets. In some cases, however, not all the words in a flit are useful to the processor. In particular, unused cache block words represent wasted power and energy. We propose to use spatial locality speculation to leverage unused words of the block transfers between the lower and upper cache levels to save energy.

1.3 Proposed Technique

The goal of the proposed technique is to reduce dynamic energy in CMP interconnect by leveraging spatial locality speculation on the expected used words in fetched cache blocks in CMP processor memory systems.

The paper makes the following contributions:

- A novel intra-cache-block spatial locality predictor, to identify words unlikely to be used before the block is evicted.
- A static packet encoding technique which leverages spatial locality prediction to reduce the network activity factor, and hence dynamic energy, in the NoC routers and links. The static encoding requires no modification to the NoC and minimal additions to the processor caches to achieve significant energy savings with negligible performance overhead.
- A complementary dynamic packet encoding technique which facilitates additional energy savings in NoC links and routers via light-weight microarchitectural enhancements.

In a 16-core CMP implemented in a 45-nm process technology, the proposed technique achieves an average of ∼35% savings in total dynamic interconnect energy at the cost of less than 1% increase in memory system latency.

The rest of this paper is organized as follows: Section 2 discusses the related work and background in caches, NoCs and power efficiency on-chip to provide the intuition behind our power saving flit-encoding technique. Section 3 discusses our proposed technique in detail, including the proposed spatial locality predictor and the proposed packet encoding schemes. Section 4 and 5 present the experimental setup and the results. Finally, we conclude in Section 6.

2 Background and Related Work

2.1 Dynamic Power Consumption

When a bit is transmitted over interconnect wire or stored in an SRAM cell, dynamic power is consumed as a result of a capacitive load being charged up and also due to transient currents during the momentary short from Vdd to Gnd while transistors are switching. Dynamic power is not consumed in the absence of switching activity. Equation 1 shows the dynamic and short-circuit components of power consumption in a CMOS circuit.

$$P = \alpha \cdot C \cdot V^2 \cdot f + t \cdot \alpha \cdot V \cdot I_{short} \cdot f \qquad (1)$$

In the equation, P is the power consumed, C is the switched capacitance, V is the supplied voltage, and F is the clock frequency. α represents the activity factor, which is the probability that the capacitive load C is charged in a given cycle. C, V, and F are a function of technology and design parameters. In systems that support dynamic voltage-frequency scaling (DVFS), V and F might be tunable at run time; however, dynamic voltage and frequency adjustments typically cannot be done at a fine spatial or temporal granularity [11]. In this work, we target the activity factor, α, as it enables dynamic energy reduction at a very fine granularity.

2.2 NoC Power and Energy

Researchers have recently begun focusing on the energy and power in NoCs, which have been shown to be significant contributors to overall chip power and energy consumption [3], [4], [12], [13].

One effective way to reduce NoC power consumption is to reduce the amount of data sent over the network. To that extent, recent work has focused on compression at the cache and network levels [14], [15] as an effective power-reduction technique. In general, however, compression techniques have overheads in terms of latency for compression and decompression. The technique we present is orthogonal to, and could potentially be used in conjunction with, these loss-less compression techniques to further reduce power. Our work seeks to reduce the amount of data transmitted through identification and removal of useless words; traditional compression could be used to more densely pack the remaining data.

Researchers have also proposed a variety of techniques to reduce interconnect energy consumption through reduced voltage swing [16]. Schinkel et al. propose a scheme which uses a capacitative transmitter to lower the signal swing to 125 mV without the use of an additional low-voltage power supply [17]. In this work we evaluate our prediction and packet encoding techniques for links composed of both full-signal swing as well as low-signal swing wires.

NoC router microarchitectures for low power have also been explored to reduce power in the transmission of data which is much smaller than a flit. Das et al. propose a novel crossbar and arbiter design that supports concurrent transfers of multiple flits on a single link to improve bandwidth utilization. [18].

Finally, static power consumption due to leakage currents is also a significant contributor to total system power. However, researchers have shown that power-gating techniques can be comprehensively applied at the NoC level and are highly effective at reducing leakage power at periods of low network activity [19].

2.3 Spatial Locality and Cache Block Utilization

Spatial and temporal locality have been studied extensively since caches came into wide use in the early 1980's [20]. Several works in the 1990's and early 2000's focused on indirectly improving spatial locality through compile and run-time program and data transformations which improve the utilization of cache lines [21]–[24]. While these techniques are promising, they either require compiler transformations or program changes and cannot be retrofitted onto existing code. Our proposed

approach relies on low-overhead hardware mechanisms and is completely transparent to software.

Hardware techniques to minimize data transfer among caches and main memory have also been explored in the literature. Sectored caches were proposed to reduce the data transmitted for large cache block sizes while keeping overhead minimal [25]. With the sectored cache, only a portion of the block (a sector) is fetched, significantly reducing both the miss time and the bus traffic. The proposed technique builds upon this idea by speculatively fetching, not just the missing sector but sectors (words in this case) which have predicted spatial locality with the miss.

Prefetching is a technique where cache lines expected to be used in the future are fetched prior to their demand request, to improve performance by reducing misses. This may come at the cost of more power spent in the interconnect between caches when inaccurate prefetches lead to unused cache block fetches. Our technique is complementary and can be used to compensate prefetch energy overhead by gating unused words. prefetches.

Pujara et al. examined the utilization of cache lines and showed that only 57% of words are actually used by the processor and the usage pattern is quite predictable [7]. They leverage this information to lower power in the cache itself by reducing the number of words read from the lower level cache and written to the upper level cache. This mechanism is orthogonal and potentially complementary to our technique, as we focus primarily on achieving lower energy consumption in the interconnect. Yoon et al. proposed an architecture that adaptively chooses memory system granularity based on spatial locality and error-tolerance tradeoffs [26]. While this work focuses on contention for off-chip memory bandwidth, our work targets on-chip interconnect energy consumption by observing spatial locality. Qureshi et al. suggested a method to pack the used words in a part of the cache after evicting it from the normal cache, thus increasing performance and reducing misses [27]. Their work thus focuses on performance rather than energy-efficiency and targets the effectiveness of the second-level cache.

Spatial locality prediction is similar to dead block prediction [28]. A dead block predictor predicts whether a cache block will be used again before it is evicted. The spatial locality predictor introduced in this paper can be thought of as a similar device at a finer granularity. The spatial locality predictor, however, takes into account locality relative to the critical word offset, unlike dead block predictors. Chen et al. predicted a spatial pattern using a pattern history table which can be referenced by the pc appended with the critical offset [29]. The number of entries in the pattern history table, as well as the number of indexes increase the memory requirement of the technique. Unlike these schemes, our predictor uses a different mechanism for managing prediction thresholds in the face of mispredictions. Kim et al. proposed spatial locality speculation to reduce energy in the interconnect [30], we present here an extended journal version of this earlier work.

3 Description

Our goal is to save dynamic energy in the memory system interconnect by eliminating switching activity associated with unused words in cache blocks transferred between the different levels of the on-chip cache hierarchy. To this end we developed a simple, low complexity, spatial locality predictor, which identifies the words expected to be used in each cache block. A used word prediction is made on a L1 cache miss, before generating a request to the L2. This prediction is used to generate the response packet eliding the unused words with the proposed flit encoding schemes described below.

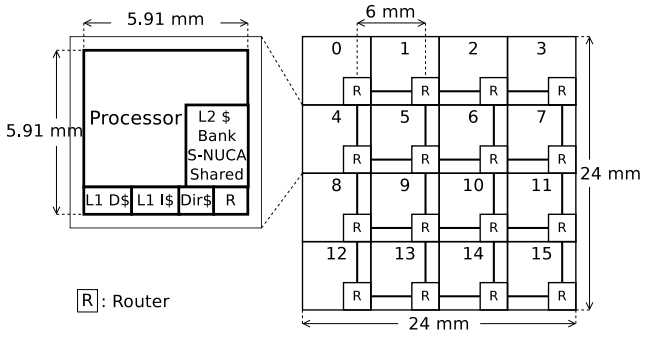


Fig. 2. General CMP Architecture

Figure 2 depicts the general, baseline architecture, representing a 16-node NoC-connected CMP. A tile consists of a processor, a portion of the cache hierarchy and a Network Interface Controller (NIC), and is bound to a router in the interconnection network. Each processor tile contains private L1 instruction and data caches. We assume the L2 is organized as a shared S-NUCA cache [31], each tile containing one bank of the L2. The chip integrates two memory controllers, accessed via the east port of node 7 and west port of node 8. Caches have a 64-byte block size. The NoC link width is 16 bytes, discounting flow-control overheads. Thus, cache-block-bearing packets are five flits long, with one header flit and four data flits. Each data flit contains four 32-bit words, as shown in Figure 5(b).

3.1 Spatial Locality Prediction

3.1.1 Prediction Overview

Our predictor leverages the history of use patterns within cache blocks brought by a certain instruction has been accessed. The intuition behind our predictor is that a given set of instructions may access multiple different memory address regions in a similar manner. In fact, we have observed that patterns of spatial locality are highly correlated to the address of the instruction responsible for filling the cache (the *fill PC*). The literature also shows that a small number of instructions cause the most cache misses [32]. Moreover, a given sequence of memory instructions accesses the same fields of data structures throughout memory [29]. Data structure instances, are unfortunately not aligned to the cache block, this misalignment can be adjusted by using the offset of the word which causes the cache miss (the *critical word offset*) while accessing the prediction table as in [7].

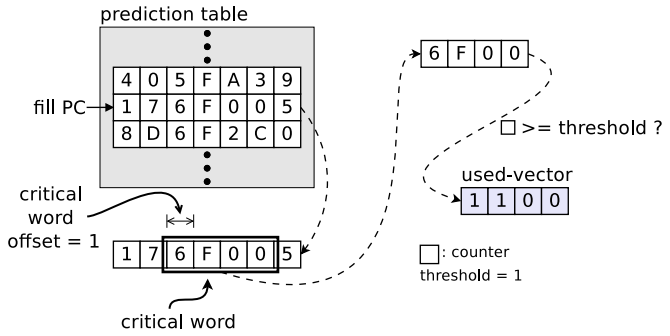


Fig. 3. Prediction example for 4 words/block cache model

3.1.2 Predictor Implementation

Our prediction table is composed of rows of four-bit saturating counters where each counter corresponds to a word in the cache block. The table is accessed such that the *fill PC* picks the row of the table, and then *n* consecutive counters starting from the *critical word offset* are chosen where *n* is the number of words in a cache block. (Thus, there are *2n - 1* counters per row to account for all possible *n - 1* offsets.) These counters represent the history of word usage in cache blocks brought by a certain memory instruction.

The value of the saturating counter relates to the probability that the corresponding word is used. The lower the counter is, the higher confidence the word will not be used. Initially, all counters are set to their max value, representing a prediction where all words are used. As cache lines are evicted with unused words, counters associated with those unused words are decremented while counters associated with used words are incremented. If a given word counter is equal to or greater than a fixed *threshold* (configured at design time), then the word is predicted to be used; otherwise, it is predicted not used. We define *used-vector* as a bit vector which identifies the words predicted used by the predictor in the cache block to be filled. A used-vector of 0xFFFF represents a prediction that all sixteen words will be used while a used-vector of 0xFF00 signifies that only the first eight words will be used.

Figure 3 shows the steps to the prediction. In this example, the number of words in a block is assumed to be 4 and the threshold is 1, for simplicity. In the figure a cache miss occurs on an instruction accessing the second word in a given block (*critical word offset = 1*). The lower-order bits of the fill PC select a row in the prediction table. Among the counters in the row, the selection window of 4 counters, which initially includes the four rightmost counters, moves to the left by the number of the critical word offset. Those selected counters are translated into a predicted used-vector based on the threshold value. The used-vector, `1100`, indicates that the first and the second words in this block will be used.

The L1 cache keeps track of the actual used vector while the block is live, as well as the lower-order bits of the fill PC, and the critical word offset. When the block is evicted from the L1, the predictor is updated with the actual used vector; if a word was used, then the corresponding counter is incremented; otherwise, it is decremented. While updating, it finds the corresponding counters with the fill-PC and the critical word offset as it does for prediction. In the event a word is falsely predicted "unused", the counters for the entire row are reset to 0xF to reduce the likelihood of future mispredictions. This form of resetting have been shown to improve confidence over up/down counters for branch predictors [33]; in initial development of the predictor we found a considerable improvement in accuracy using this technique as well. Resetting the counters allows the predictor to quickly react in the event of destructive interference and/or new program phases.

3.1.3 Impact on Energy

We model a cache with 64B blocks and 4B words. Each row of the predictor is composed of 31 four-bit saturating counters where all counters are initialized to 0xF. The predictor table has 256 rows of 31×4 bits each, thus requiring $\sim$4KB of storage. We note, although the "word" size here is 4B, this represents the prediction granularity, it does not preclude use in a 64b (8B) word architecture.

In addition to the 4KB prediction table, our scheme requires extended metadata in each cache tag. In L1, 16 bits (one per word) are necessary to determine which words have been accessed so that we can update the predictor. 8 bits for *fill-PC* and 4 bits for *critical word offset* are required to access the prediction table, as well. We also replace the single valid bit with a vector of 16 bits in L1 and L2 caches. Although, this metadata increases the power per access of the L1 and L2 caches by 0.35% and 0.72%, respectively, we also reduce the number of words read from the lower level caches and written to the upper level cache by $\sim$30% and also the number of words written back into the lower level cache by $\sim$40%. Altogether this results an average $\sim$20% reduction in dynamic energy consumption per cache access. The dynamic energy consumed by the prediction tables is discussed in Section 4.

Although not the focus of this work, leakage energy dissipation can be also optimized with the help of spatial locality speculation. Chen et al. achieved 41% of leakage energy reduction with their proposed spatial locality predictor and a circuit level selective sub-blocking technique [29]. We expect a better reduction could be achieved with our technique (as our predictor accuracy is higher), we plan to explore this in a future work on used word prediction for cache power reduction.

3.1.4 Impact on Performance

When an L1 cache miss is discovered, the predictor supplies a prediction to inform flit composition. The prediction will take two cycles: one for the table access and one for thresholding and shifting. The fastest approach would be to speculatively assume that every cache access will result in a miss and begin the prediction simultaneously with address translation; thus, the latency can be completely hidden. A more energy efficient approach is to begin the prediction as soon as the tag mismatch is discovered and simultaneously with victim selection in the L1 cache. While this approach would add a cycle to the L1 miss time, no time would be added to the more

performance critical L1 hit time. The latter approach was used in our experiments. If a word predicted unused actually is used, it is treated as a miss and all words initially predicted as unused are brought into the upper-level cache in order to correct this misprediction. This performance impact of these extra misses is discussed in Section 5.1.

On eviction of a L1 cache block, the used-vector and fill PC collected for that block are used to update the predictor. This process is less latency sensitive than prediction since the predictor does not need to be updated immediately to provide good accuracy.

3.2 Packet Composition

Once we have predicted the application's expected spatial locality to determine the unused words in a missing cache block, we employ a flit encoding technique which leverages unused words to reduce dynamic link and router energy in the interconnect between the L1, directory, L2 cache banks and memory controllers. We propose two complementary means to leverage spatial locality prediction to reduce α, the activity factor, in the NoC, thereby directly reducing dynamic energy: 1) Remove flits from NoC packets (*flit-drop*); 2) Keep unused interconnect wires at fixed polarity during packet traversal (*word-repeat*). For example, if two flits must be transmitted and all the words in the second flit are predicted unused, our *flit-drop* scheme would discard the unused flit to reduce the number of flits transmitted over the wire. In contrast, our *word-repeat* scheme would retransmit the first flit, keeping the wires at fixed polarity to reduce gate switching. These encoding schemes are also used for writeback packets to include dirty words only.

The packet compositioning may be implemented either "statically", whereby packet encoding occurs at packet generation time, or "dynamically", in which the unused words in each flit are gated within the router FIFOs, crossbars and links to avoid causing bit transitions regardless of the flits which proceed or follow it. We will first discuss the "static" packet compositioning techniques including *flit-drop*, *static-word-repeat* and their combination. We then discuss the "dynamic" packet composition techniques which allow greater reductions in activity factor, at the cost of a small increase in logical complexity in the routers and a slight increase in link bit-width.

3.2.1 Static Packet Composition

Figure 4 depicts the format of cache request and reply packet flits in our design. A packet is composed either of a head flit and a number of body flits (when the packet contains a cache block) or it consists of one atomic flit, as in the case of a request packet or a coherence protocol message. The head/atomic flit contains a used-vector. The head flit also contains source and destination node identifiers, and the physical memory address of the cache block. The remaining bytes in the head/atomic flit are unused. We assume a flow-control overhead of three bits, 1 bit for virtual channel id (VC) and 2 bits for flit type (FT). As each of body/tail flit contains data of four words (16 bytes), a flit is 16 bytes and 3 bits wide including flow control overheads.

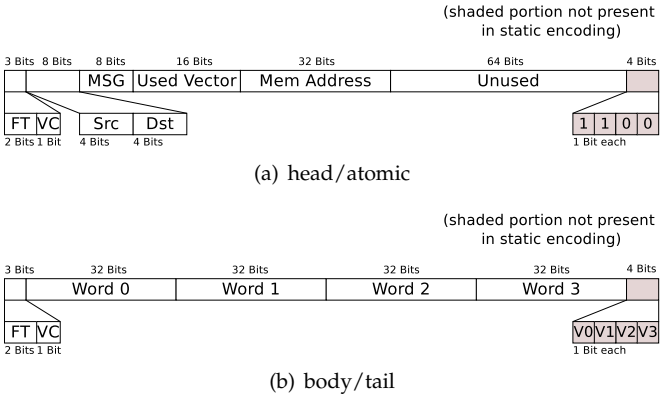


Fig. 4. Flit format for static and dynamic encoding. (Shaded portion not present in static encoding.)

Figure 5(a) depicts an example of read request (L1 fill). In this example, tile #1 requests a block at address 0x00001200 which resides in the S-NUCA L2 cache bank in tile #8. The used-vector is `1111 1100 0000 1010`, indicating the words $word_0$ - $word_5$, $word_{12}$ and $word_{14}$ are predicted used. The corresponding response packet must contain at least those words. Since the baseline architecture sends the whole block as it is, the packet contains all of the words from $word_0$ to $word_{15}$, as shown in figure 5(b).

Flit-drop: In the *flit-drop* technique, flits which are predicted to contain only unused words are dropped from the packet and only those flits which contain one or more used words are transmitted. The reduction in the number of flits per packet, reduces the number of bit transitions over interconnect wires and therefore the energy consumed. Latency due to packet serialization and NoC bandwidth will also be reduced as well. Although a read request packet may have an arbitrary used-vector, the response packet must contain all flits which have any words predicted used leading to some lost opportunity for packets which have used and unused words intermingled throughout.

Figure 5(c) depicts the response packet to the request shown in Figure 5(a) for the *flit-drop* scheme. The first body flit, containing $word_0$ - $word_3$, therefore must be in the packet as all of these words are used. The second body flit, with $word_4$ - $word_7$, also contains all valid words, despite the prediction $word_6$ and $word_7$ would not be used. These extra words are overhead in the *flit-drop* scheme because they are not predicted used but must be sent nevertheless. Although these words waste dynamic power when the prediction is correct, they may reduce the miss-prediction probability.

Static-word-repeat:The *static-word-repeat* scheme, reduces the activity factor of flits containing unused words by repeating the contents in previous flit in the place of unused words. Flits with fewer used words consume less power because there are fewer bit transitions between flits. Words marked as "used" in the used-vector contain real, valid data. Words marked as "unused" in the used-vector contain repeats of the word in the same location

FT	SRC		MSG		Mem Address	Unused
ATOM	1	8	REQ	0xFC0A	0x1200	X
	DST		Used Vector			0XFC0A = 1111 1100 0000 1010

(a) Request

HEAD	8	1	RESP	0xFFFF	0x1200	UNUSED 0	UNUSED 1
BODY0	WORD 0				WORD 1	WORD 2	WORD 3
BODY1	WORD 4				WORD 5	WORD 6	WORD 7
BODY2	WORD 8				WORD 9	WORD 10	WORD 11
TAIL	WORD 12				WORD 13	WORD 14	WORD 15

0xFFFF = 1111 1111 1111 1111

(b) Baseline Response

HEAD	8	1	RESP	0xFF0F	0x1200	UNUSED 0	UNUSED 1
BODY0	WORD 0				WORD 1	WORD 2	WORD 3
BODY1	WORD 4				WORD 5	WORD 6	WORD 7
TAIL	WORD 12				WORD 13	WORD 14	WORD 15

0xFF0F = 1111 1111 0000 1111

(c) Flit-Drop Response

HEAD	8	1	RESP	0xFC0A	0x1200	WORD 2	WORD 3
BODY0	WORD 0				WORD 1	WORD 2	WORD 3
BODY1	WORD 4				WORD 5	WORD 2	WORD 3
BODY2	WORD 4				WORD 5	WORD 2	WORD 3
TAIL	WORD 12				WORD 5	WORD 14	WORD 3

0xFC0A = 1111 1100 0000 1010

(d) Word-Repeat Response

Fig. 5. Read Request and Corresponding Response Packets (VC is not shown in this figure.)

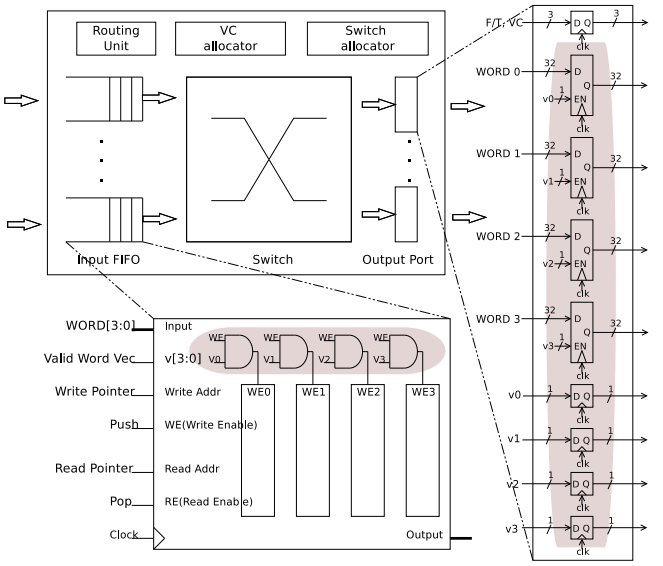


Fig. 6. Dynamic packet compositioning router. (Shaded portion not present in baseline router.)

in the previous flit. For instance, if $word_{4x+1}$ is predicted unused, the NIC places $word_{4(x-1)+1}$ in its place. As the bit-lines repeat the same bits, there are no transitions on those wires and no dynamic energy consumption. A buffer retaining four words previously fetched by the NIC is placed between the cache and the NIC and helps the NIC in repeating words. An extra mux and logic gates are also necessary in the NIC to encode repeated words.

Figure 5(d) depicts the response packet for the request in Figure 5(a) using the *static-word-repeat* scheme. In $body_1$, $word_6$ and $word_7$ are unused and, thus, replaced with $word_2$ and $word_3$ which are at the same location in the previous flit. All of the words in $body_2$ are repeated by the words in $body_1$, thus it carries virtually nothing but flow-control overhead. We also encode the unused header words, if possible.

3.2.2 Dynamic Packet Composition

The effectiveness of static packet compositioning schemes is reduced in two commonly-occurring scenarios: (a) when single-flit, atomic packets are being transmitted, and (b) when flits from multiple packets are interleaved in the channel. In both cases, repeated words in the flits cannot be statically leveraged to eliminate switching activity in the corresponding parts of the datapath. In response, we propose *dynamic packet composition* to reduce NoC switching activity by taking advantage of invalid words on a flit-by-flit basis. The difference between dynamic and static composition schemes resides primarily in how *word-repeat* treats unused words. In static composition, the unused portion of a flit is statically set at packet injection by the NIC to minimize inter-flit switching activity, requiring no changes to the router datapath. In dynamic composition, portions of the router datapath are dynamically enabled and disabled based on the validity of each word in the flit. In effect, an invalid word causes the corresponding portion of the datapath to hold its previous value, creating the illusion of word repeat.

To facilitate dynamic composition, the "used-vector" is distributed into each flit as shown in Figure 4(b). As a result the link width must be widened by four bits to accommodate the new "valid-word-vector", where each bit indicates whether the corresponding word in that flit is valid. As the figure shows, the head flit's "valid-word-vector" is always set to `1100` because the portion which corresponds to $Word_2$ and $Word_3$ of a body/tail flit are always unused.

Dynamic packet compositioning requires some modifications to a standard NoC router to enable datapath gating in response to per-flit valid bits. Figure 6 depicts the microarchitecture of our dynamic packet compositioning router. Assuming that a whole cycle is required for a flit to traverse a link, latches are required on both sides of each link. The additional logic required for link encoding is shaded in the magnified output port. Plain D-flip-flops are replaced with enable-D-flip-flops to force the repeat of the previous flit's word when the "valid-word-vector" bit for that word is set to zero, indicating that word is not used. Alternately, if the "valid-word-vector" bit for the given word is one, the word is propagated onto the link in the following cycle, as it would in the traditional NoC router. In cases where the link traversal consumes less than a full cycle, this structure could be replaced with a tristate buffer to similar effect.

We further augment the router's input FIFO buffers with per-word write enables connected to the "valid-word-vector" as shown in Figure 6. In our design, the read and write pointer control logic in the router's input FIFOs remain unmodified; however, the SRAM array storage used to hold the flits is broken into four banks, each one word in width. The "valid-word-vector" bits would gate the valid write enables going to each of word-wide banks, disabling writes associated with

unused words in incoming flits, and saving the energy associated with those word writes. The combination of these techniques for dynamic packet composition will reduce the power and energy consumption of the NoC links and router datapath proportional to the reduction in activity factor due to the *word-repeat* and *flit-drop* of unused words.

As *flit-drop* and *word-repeat* are complementary, we will also examine their combination in the evaluation section. One alternative technique we explored packs together used words into a minimal size packet. Experimentally we found this approach produces latency and power benefits negligibly different from the combination of *flit-drop* and *word-repeat*, while our technique requires less additional hardware in packet composition, so these results are not presented. These encoding schemes also are used for writebacks by marking clean words as unused.

4 Evaluation

4.1 Baseline Architecture and Physical Implementation

Figure 2 depicts the baseline architecture, representing a 16-node NoC-connected CMP. A tile consists of a processor, a portion of the cache hierarchy and a Network Interface Controller (NIC), and is bound to a router in the interconnection network. The baseline architecture employs a 4×4 2D mesh topology with X-Y routing and wormhole flow control. Each router contains 2 VCs and each input buffer is four flits deep. In our baseline configuration we assume the tiles are $36mm^2$ with 6mm-long links between nodes. Our target technology is 45 nm.

Processor Tiles: Each $36mm^2$ tile contains an in-order processor core similar to an Intel Atom Z510 ($26mm^2$) [34], a 512KB L2 cache slice ($4mm^2$), two 32KB L1 caches ($0.65mm^2$ each) and an interconnect router ($0.078mm^2$). The remaining area is devoted to a directory cache and a NIC. Our system is composed of 16 tiles and results in $576mm^2$, approximately the size of an IBM Power7 die [35]. We used CACTI 6.0 [36] to estimate cache parameters.

The L1 caches are two-way set-associative with a 2 cycle access latency. The L2 banks are 8-way set-associative with a 15-cycle access time. The 16 L2 banks spread across the chip comprise an 8-MB S-NUCA L2 [31]. Cache lines in both L1 and L2 caches are 64B wide (16 four-byte words), except where otherwise noted. Each node also contains a slice of the directory cache, interleaved the same as the L2. Its latency is 2 cycles. The number of entries in each directory cache is equal to the number of sets in an L2 bank. We assume the latency of the main memory is 100 cycles. The MESI protocol is used by the directory to maintain cache coherence. The predictor's performance is examined with the threshold value of 1 unless stated otherwise. The NoC link width is assumed to be 128 bits wide, discounting flow-control overheads.

NoC Link Wires: NoC links require repeaters to improve delay in the presence of the growing wire RC delays due to diminishing interconnect dimensions [1]. These repeaters are major sources of channel power and area overhead. Equally problematic is their disruptive effect on floorplanning, as large swaths of space must be allocated for each repeater stage. Our analysis shows that a single, energy-optimized 6 mm link in 45 nm technology requires 13 repeater stages and dissipates over 42 mW of power for 128 bits of data at 1 Ghz.

TABLE 1
Area and Power

	baseline	static	dynamic
Area (mm^2)	0.073		0.078
Static Power (mW)	0.71		0.74
Router with full-swing link			
Dynamic Power (mW)	1.73	1.28	0.92
Total Power (mW)	2.45	1.99	1.67
Router with low-swing link			
Dynamic Power (mW)	0.62	0.46	0.33
Total Power (mW)	1.34	1.18	1.08

In this work, we consider both full-swing repeated interconnects (*full-swing links*) and an alternative design that lowers the voltage swing to reduce link power consumption (*low-swing links*). We adopt a scheme by Schinkel et al. [17] which uses a capacitative transmitter to lower the signal swing to 125 mV without the use of an additional low-voltage power supply. The scheme requires differential wires, doubling the NoC wire requirements. Our analysis shows a 3.5× energy reduction with low swing links. However, low-swing links are not as static-word-repeat friendly as much as full-swing links are. There is link energy dissipation on low-swing links, even when a bit repeats the bit ahead because of leakage currents and high sense amp power consumption on the receiver side. Thus, the repeated unused-words consume $\sim 18\%$ of what used-words do. The dynamic encoding technique fully shuts down those portions of link by power gating all components with the "valid-word-vector" bits.

Router Implementation: We synthesized both the baseline router and our dynamic encoding router on a TSMC 45nm library to an operating frequency of 1Ghz. Table 1 shows the area and power of the different router designs. Note that the baseline router and the one used in static encoding scheme are identical. The table shows the average power consumed under PARSEC traffic, simulated with the methodology described in Section 4.2. The dynamic power for each benchmark is computed by dividing the total dynamic energy consumption by the execution time, then by the number of routers. Summarizing the data, a router design supporting the proposed dynamic composition technique requires ∼7% more area, while reducing dynamic power by 46% under the loads examined over the baseline at the cost of 4.3% more leakage power.

Table 2 shows the dynamic energy consumed by a flit with a given number of words encoded as used, traversing a router and a link, with respect to the three flit composition schemes: baseline(*base*), static(*sta*) and dynamic(*dyn*) encoding. In baseline, a flit always consumes energy as if it carries four used words. In static encoding, as the number of used words decreases, flits consume less energy on routers and full-swing links.

TABLE 2
Per-Flit Dynamic Energy (pJ)

	Router			Full Swing Link			Low Swing Link		
n	base	sta	dyn	base	sta	dyn	base	sta	dyn
0		0.73	0.34		0.99	2.30		0.35	0.66
1		1.31	1.01		11.52	12.83		3.34	3.67
2	3.58	1.90	2.01	43.10	22.04	23.36	12.31	6.33	6.67
3		2.77	2.79		32.57	33.89		9.32	9.68
4		3.58	3.65		43.10	44.41		12.31	12.69

n: number of used words

Static-encoding reduces NoC energy by minimizing the number of transitions on the wires in the links and in the routers' crossbars. Dynamic-encoding further reduces router energy by gating flit buffer accesses. The four-bit, valid-word-vector in each flit controls the write enable signals of each word buffer, disabling writes associated with unused words. Similarly, it also gates low-swing links, shutting down the transceiver pair on wires associated with the unused words. CACTI 6.0 [36] was used to measure the energy consumption due to accessing the predictor; which is 10.9 pJ per access.

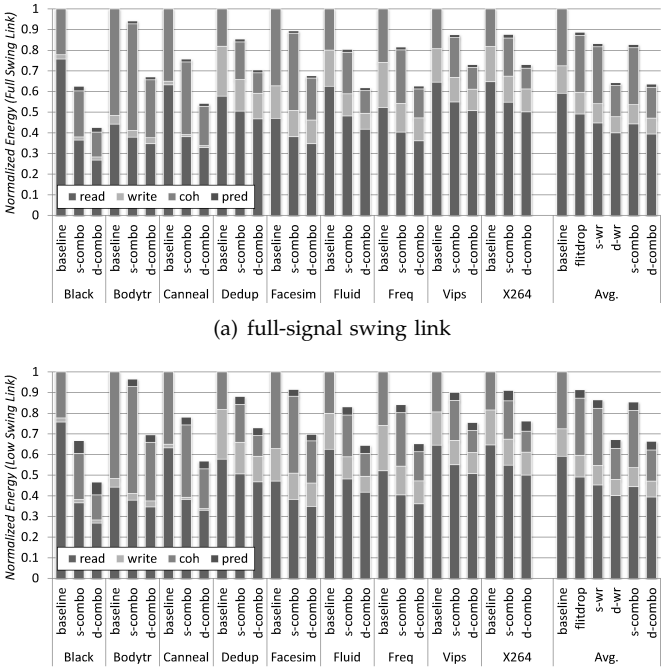
4.2 Simulation Methodology

We used the M5 full system simulator to generate CMP cache block utilization traces for multi-threaded applications [37]. Details of the system configuration are presented in section 4.1. Our workload consists of the PARSEC shared-memory multi-processor benchmarks [8], cross-compiled using the methodology described by Gebhart et. al [38]. All applications in the suite currently supported by M5 were used. Traces were taken from the "region of interest." Each trace contains up to a billion memory operations; fewer if the end of the application was reached. Cycle accurate timing estimation was performed using the *Netrace*, memory system dependence tracking methodology [39].

The total network energy consumption for each benchmark is measured by summing the energy of all L1 and L2 cache fill and spill and coherence packets as they traverse routers and links in the network. In effect, Table 2 is consulted whenever a flit with a certain number of used words traverses a router and a link. Note that even for the same flit, the used word number may vary according to the encoding scheme in use. For example, for an atomic flit, $n = 4$ in static encoding while $n = 2$ in dynamic. The predictor's energy is also added whenever the predictor is accessed.

4.3 Energy Consumption

Figure 7 shows the breakdown of dynamic energy consumption. For each benchmark, we conducted energy simulations for three configurations, each represented by one stacked bar for that benchmark: 1) *baseline* - baseline, 2) *s-combo* - static-word-repeat and flit-drop combined, and 3) *d-combo* - dynamic-word-repeat and flit-drop combined. We also show the *average* energy consumption with pure flit-drop (*flitdrop*), static-word-repeat (*s-wr*) and dynamic-word-repeat (*d-wr*). The bars are normalized against the energy consumed by baseline. Each bar is subdivided into up to four components. The first bar shows the "read" energy, energy consumed by



(a) full-signal swing link

(b) low-signal swing link

Fig. 7. Dynamic energy breakdown

cache fills and the second bar, "write", by writebacks. The third bar, "coh", shows the energy due to the cache coherence packets and the fourth bar, "pred" shows the energy consumed by predictors. The figure shows data for both full-swing and low-swing links.

In the *baseline* configuration we see that, on average, read communication consumes the most dynamic energy with ∼59% of the total. Coherence traffic consumes the second most with ∼28% of the total energy followed by write communication with ∼13% of the total energy. This breakdown follows the intuition that reads are more frequent than writes. Thus, techniques which only focus on writebacks will miss much potential gain. It is also interesting to note that cache coherence traffic shows a very significant contribution to overall cache interconnect energy. Similarly, work which does not consider atomic packets may miss significant gain.

The figure also shows that among the flit encoding schemes, *d-combo* shows the greatest improvement with ∼36% dynamic energy savings on average when full-signal swing link is used. If low-signal swing links are used, it becomes ∼34%. The pure dynamic-word-repeat (*d-wr*) is the second best resulting in additional ∼1% energy consumption. This implies that dropping flits only with flow control bits does not significantly contribute to energy reduction when dynamic- word-repeat is used. However, combining flit-drop is still beneficial to reduce latency. The combined static encoding (*s-combo*) provides an energy savings of only ∼17% and ∼15% of baseline, under full-swing and low-swing links, respectively. This indicates the significant gains that dynamic encoding provides, primarily in the cache coherence traffic which is predominately made up of single flit packets. We find the predictor merely contributes 1.5% of the total energy

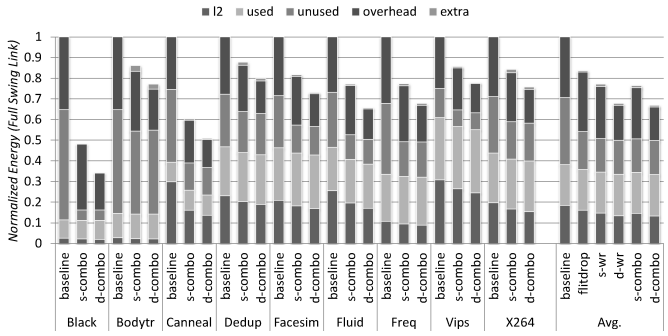


Fig. 8. Dynamic energy breakdown for reads

when full-signal swing link is used, and 4.1% when low-signal swing link is used.

Table 1 shows the average power with either type of links. It reveals that despite the increased static power, the dynamic encoding scheme still outperforms the baseline and the static encoding as well, regardless of link type.

In the following sections we will examine each of the traffic types in detail to gain a deeper understanding of the performance of our proposed technique. As full-swing link and low-swing link show similar trends, only graphs for full-swing links will be shown hereafter.

4.3.1 Read Energy Discussion

Figure 8 shows the breakdown of dynamic energy consumption for reads. Each bar is subdivided into five components and also normalized against baseline. The first bar "l2" depicts the energy consumed by L2 cache fills and spills. Although the prediction actually occurs on L1 cache misses, the encoding schemes are also used for the transactions between the L2 and memory controller, based upon used-vector generated on the L1 cache miss the lead to the L2 miss.

The second bar shows the "used" energy, energy consumed by the words which will be referenced by the program, hence "used" bars are nearly equal, with the exception of a slight increase in energy due to router overheads in the dynamic scheme. The third bar, "unused", shows the energy consumed to bring in words which will not be referenced prior to eviction. This also includes the energy consumed by words which result from *false-positive* predictions, i.e. an incorrect prediction that the word will be used. The fourth bar, "overhead", shows the energy for NoC packet overheads, including header information and flow control bits. The fifth bar, "extra", shows the energy consumed by the packet overhead due extra cache line fills to correct "false-negative" mispredictions. Our goal is to remove, as much as possible, the dynamic datapath energy consumed by unused words denoted by *unused* and, where possible, the packet overheads in *overhead*, while minimizing redundant misses due to mispredictions in *extra*. Unused words consume an average of 33% of total dynamic datapath energy, and up to 53% of total dynamic datapath energy in case of *blackscholes* (shown as *Black* in the graphs.)

The *d-combo* scheme, on average, reduces total dynamic datapath energy by ∼32%. Our prediction mechanism combined with the encoding schemes approximately halves the "unused" portion, on average. In case of *Black* where the predictor performs the best, the speculation mechanism removes 90% of the "unused" portion resulting in a 66% energy savings for cache fills when combined dynamic encoding is used. The extra transmission due to mispredictions, shown as "extra" in the stack, contributes less than 1% of energy consumption for cache fills.

4.3.2 Coherence Energy Discussion

In the simulated system coherence is maintained via the MESI protocol. Coherence protocol messages and responses represent a significant fraction of the network traffic. Those protocol messages are composed primarily of single-flit packets, and contribute ∼28% of total network energy consumption. Figure 9 shows the breakdown of dynamic energy consumption for coherence packets. Although these single-flit packets contain ∼50% unused data, as discussed in Section 3.2.1, static-encoding can not be used to reduce their energy dissipation. Dynamic-encoding, however, reduce it by up to 45.5%.

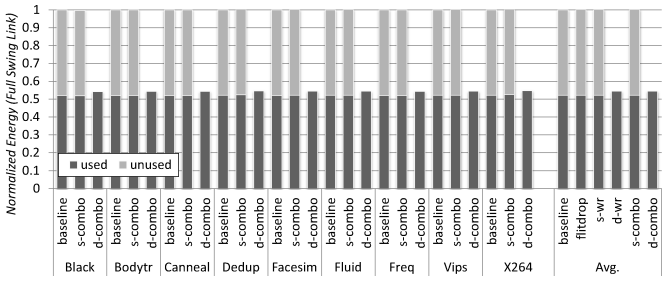


Fig. 9. Dynamic energy breakdown for coherent packets

4.3.3 Write Energy Discussion

Figure 7 shows that writebacks consume an average of 13% of total dynamic energy. Upon dirty line writeback, we find that, on average, 40% of the words in the block are clean, and those words contribute 23% of the total energy consumed by writebacks.

Figure 10 shows the dynamic energy breakdown caused by writebacks. The first bar, "dirty", shows the energy consumed by the dirty words in cache lines. The second bar "overhead" shows the energy consumed by NoC packet overheads. The third bar, "clean", includes the link energy consumed by sending clean words in writeback packets. Our goal is to remove the portion of the energy consumption associated with transmitting "clean" words. On average, the *s-combo* scheme reduces the energy consumption due to writebacks by 29%. Further savings are achieved by *d-combo*. It encodes not only body/tail flits but also head flits of the writeback packets resulting in a 40% savings. When full swing links are used, it is possible to remove all of energy dissipation due to clean words with the static flit-encoding scheme. However, when static word repeat is used with low swing links, although clean words are encoded to repeat the words in the flit ahead, those words cause energy dissipation due to leakage currents and high sense amp power consumption on the receiver side.

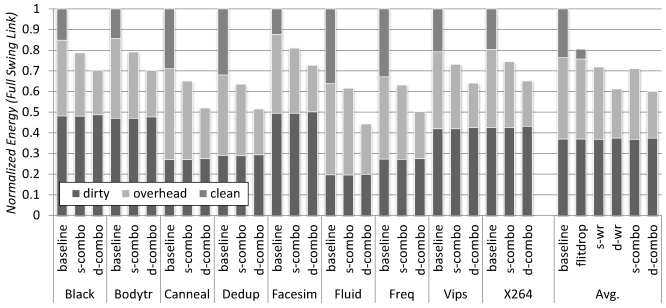
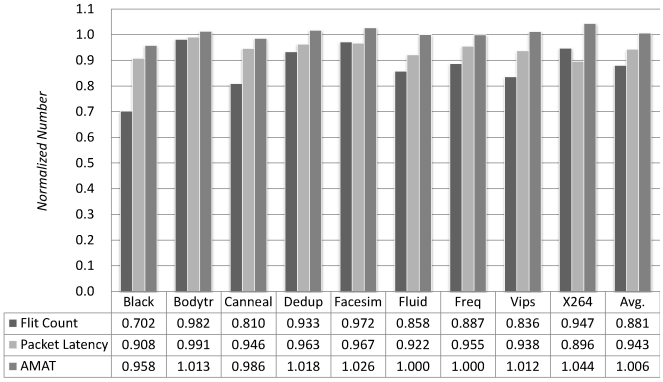


Fig. 10. Dynamic energy breakdown for writes



	Black	Bodytr	Canneal	Dedup	Facesim	Fluid	Freq	Vips	X264	Avg.
Flit Count	0.702	0.982	0.810	0.933	0.972	0.858	0.887	0.836	0.947	0.881
Packet Latency	0.908	0.991	0.946	0.963	0.967	0.922	0.955	0.938	0.896	0.943
AMAT	0.958	1.013	0.986	1.018	1.026	1.000	1.000	1.012	1.044	1.006

Fig. 11. Flit count, packet latency and AMAT normalized against baseline.

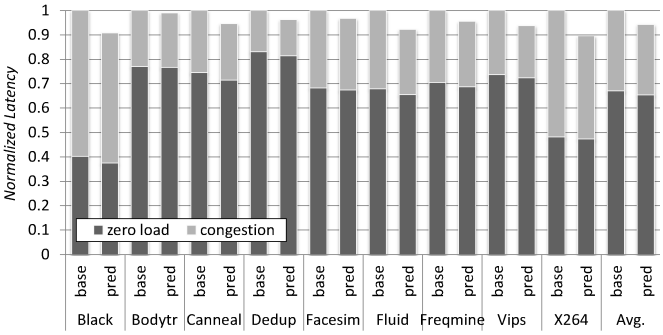


Fig. 12. Packet latency breakdown

5 Analysis

In this section we analyze the performance impact of the proposed energy reduction technique, explore predictor training and compare against a 32-byte cache line baseline design.

5.1 Performance

The performance impact of the proposed technique is governed by two divergent effects. First, the scheme should improve performance because flit-drop reduces the number of flits injected into the network. Decreased flit count improves performance through less serialization latency, and reduced congestion due to lower load. Second, offsetting the benefit from flit-drop, incorrectly predicting a word unused can lead to more misses, increasing network load and average memory access time (AMAT). To quantify the impact of the proposed technique, Figure 11 shows the reduction in flits injected into the network for each benchmark, the reduction in individual packet latency, and the AMAT for each benchmark, all normalized against baseline. Each value number is normalized against baseline. As the figure shows, although flit count and individual packet latency decrease significantly, AMAT is essentially flat across the benchmarks. In this section we examine the relationship between network performance and system performance.

5.1.1 Network Performance

As shown in Figure 11, the flit count is reduced by 12% on average. Optimally, the flit count reduction should be directly proportional to the block utilization. From Figures 1, 7 and 11, we see that Blackscholes, which has the lowest block utilization, and the greatest portion of read energy consumption, has the greatest reduction flits across the PARSEC benchmarks. Alternately, Bodytrack, which also shows one of the lowest block utilizations, removes merely 1.8% the injected flits. This is because flit count reduction is related not only to block utilization, but also prediction accuracy, proportion of single flit packets, and the used-unused pattern within the packet. In Bodytrack, flit reduction is low because single-flit coherent packets make up a larger portion of the injected packets, and the predictor is less accurate than for Blackscholes.

Lowered flit count should be correlated with reduced packet latency. Figure 11 shows normalized packet latency. On average, the network latency is reduced by ∼6% as the number of flits has decreased. In Blackscholes, with greatest reduction in flit count, the packet latency is reduced by 9%, showing one of the best network performance improvements across the PARSEC benchmarks. Alternately, Bodytrack's network performance is improved by only 1%. Interestingly, flit count reduction and network latency are not always strictly proportional to each other; X264, counter-intuitively shows a greater improvement in packet latency than its reduction in flit count, warranting further analysis.

Flit-drop improves network performance not only by reducing the serialization latency but also by avoiding network congestion. Figure 12 shows the breakdown of average packet latency. Each bar consists of two components; 1) *zero load* shows the packet latency due to static hop count and serialization latencies, 2) *congestion* shows the latencies due to the resource conflicts. Although each component contributes 67% and 33% of the average latency, respectively, the greater impact of reduced flit count lies in *congestion*. This effect is illustrated by X264 which has the second greatest congestion latency, as a result a relatively small reduction in flits translates into a greater reduction in packet latency. The overall average packet latency has been reduced by 5.7%.

5.1.2 Overall System Performance Discussion

As a proxy for overall system performance we examine the technique's impact on average memory access time (AMAT). Despite an improvement in packet latency, Figure 11 shows that AMAT is unchanged on average,

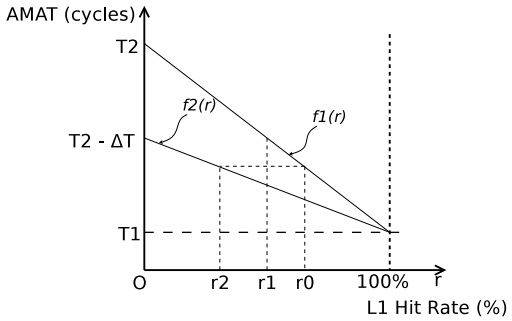


Fig. 13. AMAT graph

with some benchmarks showing a slight improvement, while others showing a slight degradation. To explore this counter-intuitive result we examine how AMAT relates to packet latency and L1 miss rate. AMAT in this work and it can be estimated by Equation 2.

$$AMAT = Latency(L1) + (1 - HitRate(L1)) \times Latency(L2+) \quad (2)$$

In this equation, $Latency(L1)$ is the constant L1 latency for the system, and $HitRate(L1)$ is the hit rate of L1 accesses which varies by benchmark locality and is effected by false unused-word predictions. $Latency(L2+)$ is the latency of memory accesses served by L2 and beyond, which is also known as L1 *miss* latency. It is a function of the constant L2 cache access time, L2 miss rate, network latency and the constant memory access time. Assuming $Latency(L2+)$ is fixed, AMAT is a linear function of $HitRate(L1)$. Figure 13 visualizes AMAT as $f(r)$ where r is L1 hit rate.

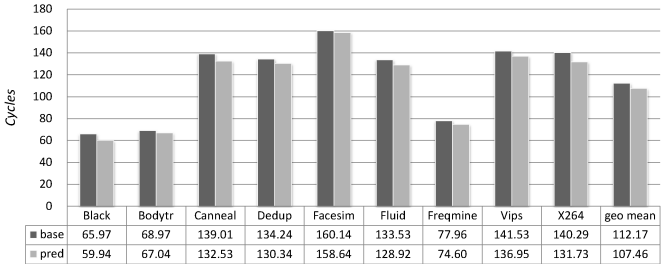
In Figure 13, let $f_1(r)$ be the AMAT characteristic for a benchmark, where T_1 is the L1 latency and T_2 is L1 latency plus L1 miss latency. Say, with the baseline scheme, the L1 hit rate is r_0 and AMAT becomes $f_1(r_0)$. If our prediction mechanism drops the L1 hit rate to r_1 and that does not change L1 miss latency, the AMAT becomes $f_1(r_1)$. In such a case, $f_1(r_1) - f_1(r_0)$ represents the performance loss due to mispredictions. However, thanks to our packet composition technique, L1 miss latency, in general, is lower than that of baseline cases. Thus, its AMAT characteristic function should be redrawn as $f_2(r)$ and the AMAT at r_1 is $f_2(r_1)$. If $f_2(r_1) < f_1(r_0)$ as in this example, the difference, $f_1(r_0) - f_2(r_1)$ denotes the performance benefit from our prediction technique.

In this example, we can also see that as long as the predictor drops the L1 hit rate no lower than r_2, performance improvement is expected. We define *safe range* as the range of L1 hit rate where our prediction scheme shows equal or better AMAT than the baseline design. In this particular example, the safe range is $[r_2, r_0]$. To generalize, safe range, Δr, is calculated as below:

$$\Delta r = \frac{\Delta T}{T_o - \Delta T}(1 - r_o) \quad (3)$$

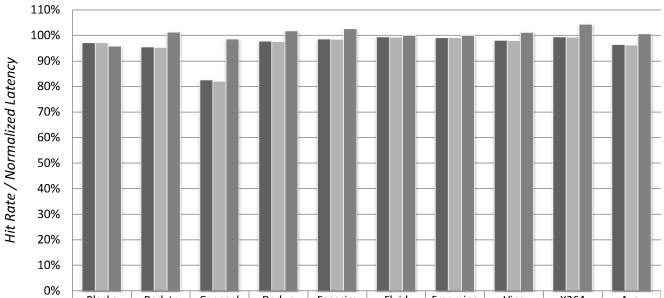
where T_o and r_o are the original L1 miss latency and L1 hit rate, respectively, and ΔT the reduced amount of L1 miss latency. The wider safe range we have, the better chance that we achieve the performance improvement.

Figure 14(a) shows the average L1 miss latency for each benchmark. The bars marked as *base* show the L1 miss latency for the baseline design, while *pred* shows the latency for the proposed scheme. In every case, a reduction in L1 miss latency is observed. This reduction is closely related to the reduction in the interconnect network latency shown in Figure 12. Although the packet latency in the figure is normalized, the L1 miss latency shows the similar trend to that; the more reduction in the network latency we have, the more reduction in L1 miss latency. According to Equation 3, the wider safe range, and, in turn, no performance degradation, is expected for benchmarks with a lower *base* and a bigger gap between *base* and *pred*. However, the safe range is still up to L1 hit rate.



	Black	Bodytr	Canneal	Dedup	Facesim	Fluid	Freqmine	Vips	X264	geo mean
base	65.97	68.97	139.01	134.24	160.14	133.53	77.96	141.53	140.29	112.17
pred	59.94	67.04	132.53	130.34	158.64	128.92	74.60	136.95	131.73	107.46

(a) L1 Miss Latency



	Blacks	Bodytr	Canneal	Dedup	Facesim	Fluid	Freqmine	Vips	X264	Avg.
l1 hit base	97.12%	95.47%	82.63%	97.75%	98.60%	99.39%	99.15%	98.12%	99.46%	96.41%
l1 hit pred	97.10%	95.24%	82.06%	97.61%	98.52%	99.37%	99.11%	98.02%	99.33%	96.26%
norm lat	0.958	1.013	0.986	1.018	1.026	1.000	1.000	1.012	1.044	1.006

(b) L1 Hit Rate

Fig. 14. Overall system performance

Figure 14(b) shows the original L1 hit rate (*"l1 hit base"*) the new L1 hit rate (*"l1 hit pred"*) and the changed average memory access time (*"norm lat"*). On average, L1 hit rate is decreased by 0.15%. Blackscholes ("Black") has the third greatest improvement in L1 miss latency, the lowest L1 miss latency and the second lowest L1 hit rate, it results in the greatest overall performance improvement of 4.2%. By contrast, "X264", while having the greatest L1 miss latency improvement, also has one of the highest L1 miss latency, and the highest L1 hit rate, therefore achieves the worst overall system performance. Although "Canneal" shows the worst impact on L1 hit rate, due to its low original L1 hit rate, the reduced L1 hit rate is still in its safe range, thus, no performance penalty for mispredictions is shown.

5.2 Predictor Tuning

As with many speculative techniques, our scheme incurs a performance penalty when mis-speculation occurs. In this case, the penalty manifests as increased L1D misses.

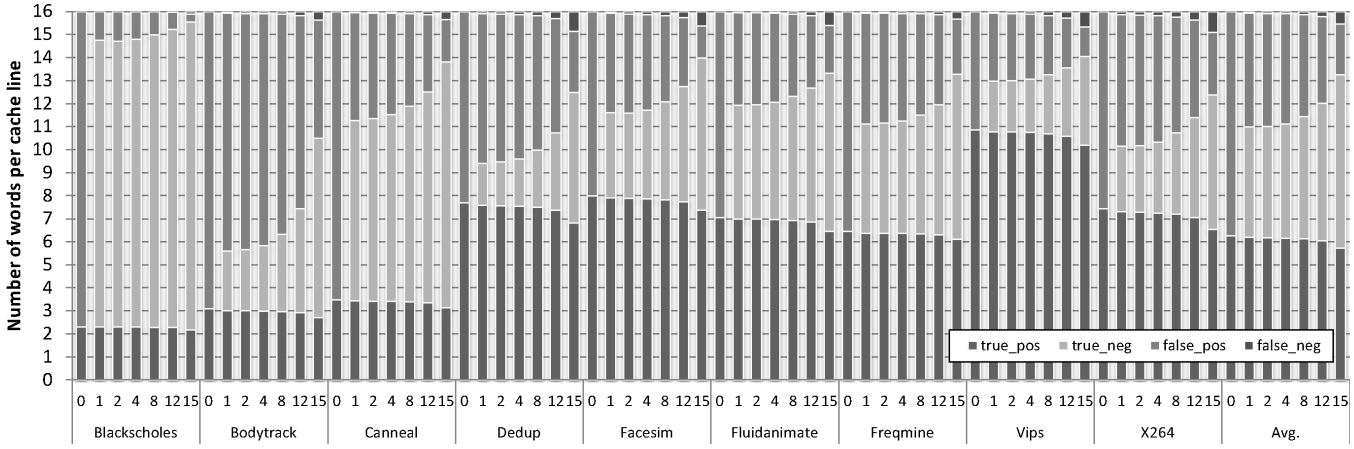


Fig. 15. Breakdown of Predictions Outcomes

As Figure 14(b) shows, L1D hit rates are barely impacted by our technique. One possible interpretation of this data is that our predictor is overly conservative, and that energy gain could be achieved by more aggressively tuning our predictor, in this section we explore predictor tuning to this end.

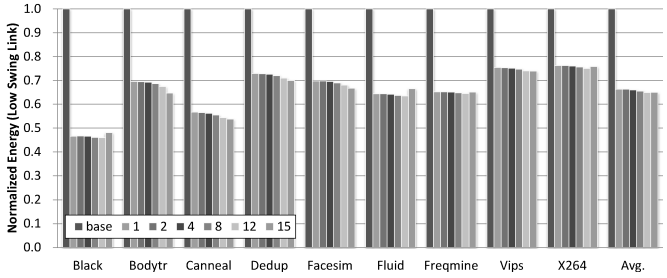
Our prediction model requires a threshold value to be configured at design time. As described in Section 3.1.1, the threshold determines whether a certain word will be used or not according to its usage history counter. If the counter value is less than the threshold, the word is predicted to be unused. The smaller threshold value, the more biased the predictor towards predicting a word will be used. Thus, the threshold value tunes the trade-off between energy consumption and memory access time.

Figure 15 shows the prediction outcomes with respect to various threshold values (numbers along the bottom) for each of the benchmarks examined. Each bar is broken into components to show average number of words in a cache line with the following characteristics. The bars marked *true_pos* show the fraction of true positives: words predicted used and actually used. The bars marked *true_neg* show the portion of true negatives: words predicted unused and actually not used. The words in this category are the source of the energy reduction in our design. These two categories form the portion of the words that the predictor correctly speculates their spatial localities. The bars marked *false_pos* show the fraction of false positives: words predicted used but actually unused. The words in this category do not cause any miss penalty but represent a lost opportunity for energy reduction. Finally, the bars marked *false_neg* show the portion of false negatives: words predicted unused but actually used. These words result in additional memory system packets, potentially increasing both energy and latency. The threshold value "0" in this figure represents the baseline configuration, where all words are assumed used. In general, as the threshold value increases, the portions of *true_neg* and *false_neg* increase while *true_pos* and *false_pos* decrease. This implies that the higher threshold chosen, the lower energy consumption (due to *true_neg* predictions) but also the higher the latency(due to *false_neg* predictions). We also note that even with the most aggressive threshold setting, a significant number of *false_pos* predictions remain, despite significant increases in *false_neg* predictions, implying that headroom for improvement via a more accurate prediction mechanism exists.

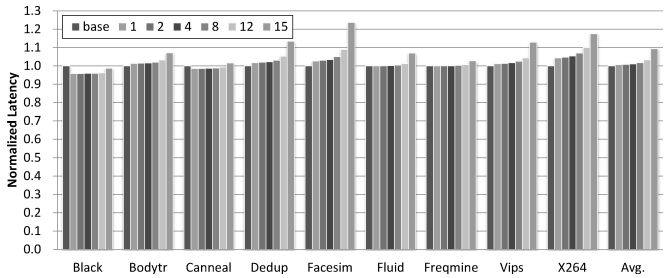
Figure 16 depicts the normalized energy consumption and normalized average memory access time (AMAT) for threshold values from 1 to 15. For this experiment, we use low swing links, a similar trend of energy consumption and AMAT was found for full swing links. Figure 16(a) shows a modest downward trend in energy consumption as the threshold value increases, with the greatest increase between thresholds of 8 and 12. This is the expected outcome of growing *true_neg* with higher threshold values in Figure 15. In some benchmarks, such as "Black", "Fluid" and "X264", there is slight increase at the highest threshold value. The main reason for this increase is the energy required to service increased L1D misses, which overcome the benefit of transmitting fewer words in the initial request.

Figure 16(b) shows the normalized AMAT with varying threshold. In general, the latency shows a modest upward trend as the threshold grows. The higher the threshold, the more words are speculated as unused by the predictor, leading to increased L1 miss rates and degrading the overall memory system latency. Though this trend becomes dramatic for a threshold of 15, increasing AMAT by up to 23% for one application; we find that thresholds of less than 4 have a minimal negative impact on AMAT.

Given our goal was to decrease energy with a minimal impact on performance, we use the $Energy \times Delay^2$ metric as a figure of merit for our design. Experimentally we determined that $Energy \times Delay^2$ is approximately equal across the thresholds between 1 and 8, however, it considerably increases beyond the threshold 12. This result validates our choice a threshold value of 1 in our experiments. We find the performance impact with this bias is negligible. On average, with this threshold, the additional latency of each operation is $\sim$0.6%. These results show that further energy savings could be achieved through improved predictor accuracy, which we leave to future work.



(a) Normalized Energy Consumption

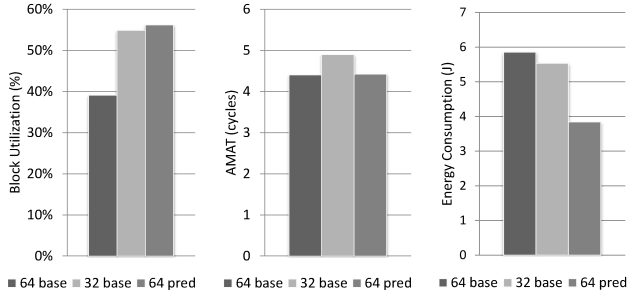


(b) Normalized Average Memory Access Time (AMAT)

Fig. 16. Normalized energy and AMAT for different threshold values

5.3 Case Study: Comparison with Smaller Lines

Naïvely, one might conclude that the low cache block utilization shown in Figure 1 could be an indication that cache line size is in-fact too long, and that utilization could be improved by implementing a smaller cache line size. To explore this concern we examine our technique versus a 32-byte cache lines baseline.



(a) Block Utilization (b) AMAT (c) Energy

Fig. 17. Comparison to a smaller cache line

Figure 17 shows results for three different configurations; 1) the baseline design with 64-byte cache lines (*64 base*), 2) a baseline design with 32-byte lines (*32 base*) keeping the cache size and associativity the same as *64 base*, and 3) 64-byte lines with our prediction and dynamic packet composition technique (*64 pred*). Figure 17(a) shows the arithmetic average of block utilization for each configuration across the PARSEC benchmarks. Figure 17(b) shows the geometric mean of AMAT and Figure 17(c) depicts the geometric mean of total energy consumption. These figures show, 32-byte lines have better utilization than the 64-byte baseline. Compared with *64 base*, however, only a marginal energy reduction is achieved at the cost of considerable performance loss. The smaller cache block size results in the increased L1 misses, and thereby, increased latency of memory accessing operations. *64 pred*, shows even greater block utilization than *32 base* while maintaining the performance of *64 pred* and consuming much less energy than the rest. Hence, the proposed technique is a better design choice than shrinking cache lines to reduce power.

6 Conclusions

In this paper, we introduce a simple, yet powerful mechanism using spatial locality speculation to identify unused cache block words. We also propose a set of static and dynamic methods of packet composition, leveraging spatial locality speculation to reduce energy consumption in CMP interconnect. These techniques combine to reduce the dynamic energy of the NoC datapath through a reduction in the number of bit transitions, reducing α the activity factor of the network.

Our results show that with only simple static packet encoding, requiring no change to typical NoC routers and very little overhead in the cache hierarchy, we achieve an average of 17% reduction in the dynamic energy of the network if full-signal swing links are used. Our dynamic compositioning technique, requiring a small amount of logic overhead in the routers, enables deeper energy savings of 36% and 34%, for full-swing and low-swing links respectively.

References

[1] International Technology Roadmap for Semiconductors (ITRS) Working Group, "International Technology Roadmap for Semiconductors (ITRS), 2009 Edition." http://www.itrs.net/Links/2009ITRS/Home2009.htm.

[2] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *The 38th International Design Automation Conference (DAC)*, 2001.

[3] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, 2007.

[4] M. Taylor, M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar Operand Networks: On-chip Interconnect for ILP in Partitioned Architectures," in *The IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2002.

[5] D. Molka, D. Hackenberg, R. Schone, and M. S. Muller, "Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System," in *The 18th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2009.

[6] Advanced Micro Devices (AMD) Inc., "AMD Opteron Processors for Servers: AMD64-Based Server Solutions for x86 Computing." http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8796,00.html.

[7] P. Pujara and A. Aggarwal, "Cache Noise Prediction," *IEEE Transactions on Computers*, vol. 57, 2008.

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *The 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[9] P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger, "Implementation and Evaluation of On-Chip Network Architectures," in *IEEE International Conference on Computer Design (ICCD)*, 2006.

[10] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. McDonald, S. W. Keckler, and D. Burger, "Implementation and Evaluation of a Dynamically Routed Processor Operand Network," in *The 1st ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, 2007.

[11] L. Shang, L.-S. Peh, and N. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," in *High-Performance Computer Architecture, 2003.*

[12] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration," in *In The Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2009.

[13] A. Banerjee, R. Mullins, and S. Moore, "A Power and Energy Exploration of Network-on-Chip Architectures," in *The First International Symposium on Networks-on-Chip*, 2007.

[14] R. Das, A. Mishra, C. Nicopoulos, D. Park, V. Narayanan, R. Iyer, M. Yousif, and C. Das, "Performance and power optimization through data compression in network-on-chip architectures," in *High Performance Computer Architecture, 2008. IEEE 14th International Symposium on*.

[15] Y. Jin, K. H. Yum, and E. J. Kim, "Adaptive data compression for high-performance low-power on-chip networks," in *The 41st annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2008.

[16] H. Zhang, V. George, and J. Rabaey, "Low-swing on-chip signaling techniques: effectiveness and robustness," *IEEE Transactions on VLSI Systems*, vol. 8, no. 3, 2000.

[17] D. Schinkel, E. Mensink, E. Klumperink, E. van Tuijl, and B. Nauta, "Low-power, high-speed transceivers for network-on-chip communication," *IEEE Transactions on VLSI Systems*, vol. 17, no. 1, 2009.

[18] R. Das, S. Eachempati, A. Mishra, V. Narayanan, and C. Das, "Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, 2009.

[19] K. Hale, B. Grot, and S. Keckler, "Segment gating for static energy reduction in networks-on-chip," in *Network on Chip Architectures, 2009. 2nd International Workshop on*.

[20] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[21] S. Carr, K. S. McKinley, and C.-W. Tseng, "Compiler Optimizations for Improving Data Locality," *SIGPLAN Notices*, vol. 29, no. 11, 1994.

[22] B. Calder, C. Krintz, S. John, and T. Austin, "Cache-Conscious Data Placement," in *The 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1998.

[23] T. M. Chilimbi, B. Davidson, and J. R. Larus, "Cache-Conscious Structure Definition," *SIGPLAN Notices*, vol. 34, no. 5, 1999.

[24] T. Chilimbi, M. Hill, and J. Larus, "Making Pointer-Based Data Structures Cache Conscious," *IEEE Computer*, vol. 33, no. 12, 2000.

[25] J. S. Liptay, "Structural aspects of the System/360 Model 85, II: The cache," *IBM Systems Journal*, vol. 7, no. 1, 1968.

[26] D. H. Yoon, M. K. Jeong, and M. Erez, "Adaptive granularity memory systems: a tradeoff between storage efficiency and throughput," in *Proceedings of the 38th annual international symposium on Computer architecture*, 2011.

[27] M. K. Qureshi, M. A. Suleman, and Y. N. Patt, "Line distillation: Increasing cache capacity by filtering unused words in cache lines," *High-Performance Computer Architecture, International Symposium on*, vol. 0, 2007.

[28] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-block prediction & dead-block correlating prefetchers," *SIGARCH Comput. Archit. News*, vol. 29, no. 2, 2001.

[29] C. F. Chen, S. hyun Yang, and B. Falsafi, "Accurate and complexity-effective spatial pattern prediction," in *In HPCA-10*, IEEE Computer Society, 2004.

[30] H. Kim, P. Ghoshal, B. Grot, P. V. Gratz, and D. A. Jimènez, "Reducing Network-on-Chip Energy Consumption Through Spatial Locality Speculation," in *The Fifth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, 2011.

[31] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches," in *ACM SIGPLAN NOTICES*, 2002.

[32] S. G. Abraham, R. A. Sugumar, D. Windheiser, B. R. Rau, and R. Gupta, "Predictability of load/store instruction latencies," in *Proceedings of the 26th annual international symposium on Microarchitecture*, IEEE Computer Society Press, 1993.

[33] E. Jacobsen, E. Rotenberg, and J. E. Smith, "Assigning confidence to conditional branch predictions," in *Proceedings of the 29th Annual International Symposium on Microarchitecture*, 1996.

[34] Intel, "Intel Atom Processor Z510." http://ark.intel.com/Product.aspx?id=35469&processor=Z510&spec-codes=SLB2C.

[35] Jon Stokes, "IBM's 8-core POWER7: twice the muscle, half the transistors." http://arstechnica.com/hardware/news/2009/09/ibms-8-core-power7-twice-the-muscle-half-the-transistors.ars.

[36] 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 Annual IEEE/ACM International Symposium on Microarchitecture*, (Washington, DC, USA), IEEE Computer Society, 2007.

[37] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, 2006.

[38] M. Gebhart, J. Hestness, E. Fatehi, P. Gratz, and S. W. Keckler, "Running PARSEC 2.1 on M5," tech. rep., The Univ. of Texas at Austin, Dept. of Comp. Sci., 2009.

[39] J. Hestness and S. W. Keckler, "Netrace: Dependency-tracking traces for efficient network-on-chip experimentation," tech. rep., The Univ. of Texas at Austin, Dept. of Comp. Sci., 2011.

Hyungjun Kim is a Ph.D candidate in the department of Electrical and Computer Engineering at Texas A&M University. He received his B.S. degree from Yonsei University in 2002, and his M.S. degree in Electrical Engineering from University of Southern California in 2008. He is currently working with Professor Paul V. Gratz. His research interests include low power memory systems and on-chip interconnection networks.

Boris Grot is a postdoctoral researcher at EPFL. His research interests include processor architectures, memory systems, and interconnection networks for high-throughput, energy-aware computing. Grot has a PhD in computer science from the University of Texas at Austin. He is a member of IEEE and the ACM.

Paul V. Gratz (S'04-M'09) is an Assistant Professor in the department of Electrical and Computer Engineering at Texas A&M University. He received his B.S. degree, and M.S. degrees in Electrical Engineering from The University of Florida in 1994 and 1997 respectively. He received his Ph.D. degree in Electrical and Computer Engineering from the University of Texas at Austin in 2008. His research interests include high performance computer architecture, processor memory systems and on-chip interconnection networks. He is a member of the IEEE and ACM.

Daniel A. Jiménez is Professor and Chair of the Department of Computer Science at the University of Texas at San Antonio. His research interests include microarchitecture and low-level compiler optimizations. Jiménez has a PhD in Computer Sciences from the University of Texas at Austin. He is a Member of the IEEE and a Senior Member of the ACM.