Reducing Network-on-Chip Energy Consumption Through Spatial Locality Speculation

Hyungjun Kim* Pritha Ghoshal* Boris Grot† Paul V. Gratz* Daniel A. Jiménez‡
*Department of Electrical and Computer Engineering, Texas A&M University
†Department of Computer Science, The University of Texas at Austin
‡Department of Computer Science, The University of Texas at San Antonio
{hyungjun, pritha9987, pgratz}@tamu.edu, bgrot@cs.utexas.edu, 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 misprediction rate of just 2.5%; (b) the combined NoC energy savings enabled by the predictor and microarchitectural support are 35% on average and up to 60% in the best case; and (c) the performance impact of these energy optimizations is negligible.

Categories and Subject Descriptors

B.3.0 [**MEMORY STRUCTURES**]: General

General Terms

Design, Measurement

Keywords

Cache Design, Flit Encoding

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 monolithic processors 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.

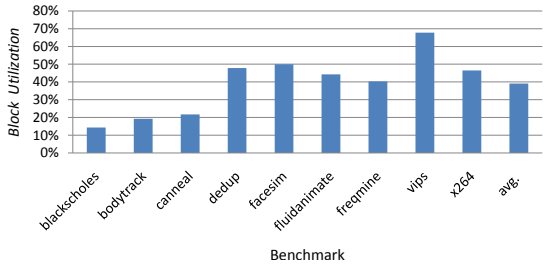


Figure 1: Percentage of cache block words utilized by applications in the PARSEC multithreaded benchmark suite.

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 some degree of 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].

Figure 1 shows the percentage of words utilized in applications from 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. 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 data that ultimately is not used.

1.2 Proposed Technique

The goal of the proposed technique is to reduce dynamic energy consumption 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:

NOCS '11 May 1-4, 2011 Pittsburgh, PA, USA

- 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 transmitted flits, reducing switching activity in NoC links and routers via light-weight microarchitectural support.

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.

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.

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

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 [9]. 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, 10, 11].

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 [12, 13] as an effective power-reduction technique. Compression is complementary to our approach. While our work seeks to reduce the amount of data transmitted through identification of useless words, 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 [14]. 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 [15]. 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.

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 [16].

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 [17]. Several works in the 1990's and early 2000's focused on indirectly improving spatial locality through compile and runtime program and data transformations which improve the utilization of cache lines [18–21]. 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 [22]. 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.

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 through power gating of portions of the cache array. This mechanism is orthogonal and potentially complementary to our technique, as we focus primarily on achieving lower energy consumption in the interconnect. 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 [23]. 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 [24]. 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 [25]. 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.

3. DESCRIPTION

Our goal is to save dynamic energy in the memory system interconnect NoC. 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 the request packet to the L2 is generated. This spatial locality predic-

tion is used to reduce dynamic energy consumption in the NoC within the cache hierarchy.

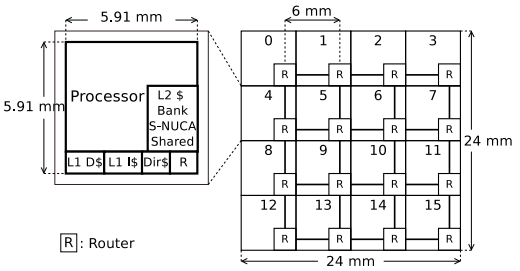


Figure 2: General CMP Architecture

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 [26], 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 5b.

3.1 Spatial Locality Prediction

When a block must be filled in the L1 cache, a spatial locality predictor is consulted to generate a prediction of the words likely to be used. We define *used-vector* as a bit string 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.

3.1.1 Prediction Overview

The principle of our predictor stems from the intuition that a set of instructions may access several memory blocks 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*). We have also observed that in order to avoid aliasing and to achieve better accuracy in the prediction, the offset of the word which causes the cache miss (the *critical word offset*) is also necessary for the predictor. Thus, our predictor takes those two inputs to make the prediction.

The prediction is based on the history of patterns in which cache blocks brought by a certain instruction has been accessed. The history of patterns is represented by a row of four-bit saturating counters where each counter corresponds to a word in the cache block. Our prediction table is composed of rows of these counters. The value of the saturating counter is proportional to the probability that a corresponding word is used. The lower the counter is, the higher confidence that 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 exceeds a fixed

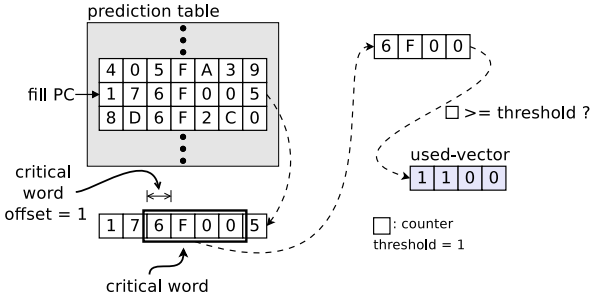


Figure 3: Prediction Example

threshold (configured at design time), then the word is predicted to be used; otherwise, it is predicted not 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 configured as 1. The cache miss occurs because an instruction is accessing the second word in the missing 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. Thus, the number of counters per row must be twice the number of words per block, less one to account for all possible critical word offsets. Then, those selected counters are translated into the 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. Resetting counters have been shown to improve confidence over up/down counters for branch predictors [27]; we measure a considerable improvement in accuracy using this technique as well.

3.1.2 Predictor Implementation

We model a cache with 64b blocks and 4B words, which requires used-vectors of 16 bits (one per word) for tracking used words in a block. 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 these architectures, data accesses come in a range of sizes from 1B up to 8B; 8B accesses would be represented with two used "words" while 4B and less would be represented by one.

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.

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 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 re-transmit the first flit, keeping the wires at fixed polarity to reduce gate switching.

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

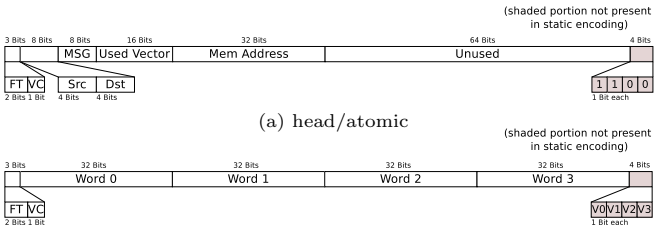


Figure 4: Flit format for static and dynamic encoding. (Shaded portion not present in static encoding.)

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.

Figure 5a 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 5b.

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 5c depicts the response packet to the request shown in Figure 5a 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.

FT	SRC	MSG	Used Vector	Mem Address	Unused
ATOM	1	8	REQ	0xFC0A	0x1200	X
	DST				

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

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

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 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 previ-

ously 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 5d depicts the response packet for the request in Figure 5a 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

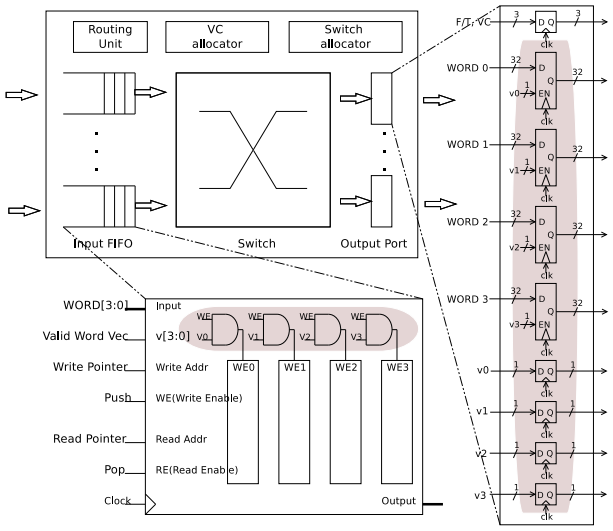


Figure 6: Dynamic Packet Compositioning Router

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 4b. 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 examine their combination in the evaluation section. 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 ($26mm^2$) [28], 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 [29].

We used CACTI 6.0 [30] 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 [26]. Cache lines in both L1 and L2 caches are 64B wide (16 four-byte words). 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

Table 1: Area and Power

	baseline	static	dynamic
Area (mm^2)	0.073		0.078
Static Power (mW)	0.71		0.75
Router with full-swing link			
Dynamic Power (mW)	1.47	1.02	0.78
Total Power (mW)	2.18	1.73	1.53
Router with low-swing link			
Dynamic Power (mW)	0.54	0.38	0.28
Total Power (mW)	1.25	1.09	1.03

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.

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. [15] 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 $\sim 20\%$ 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 using 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 $\sim$7% more area, while reducing dynamic power by 47% under the loads examined over the baseline at the cost of 5.6% more leakage power.

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	3.58	0.73	0.34	43.10	0.99	2.30	12.31	0.35	0.66
1	3.58	1.31	1.01	43.10	11.52	12.83	12.31	3.34	3.67
2	3.58	1.90	2.01	43.10	22.04	23.36	12.31	6.33	6.67
3	3.58	2.77	2.79	43.10	32.57	33.89	12.31	9.32	9.68
4	3.58	3.58	3.65	43.10	43.10	44.41	12.31	12.31	12.69

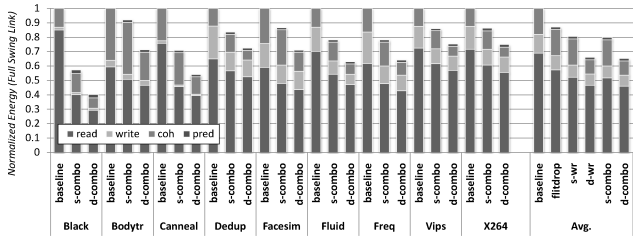
n: number of used words

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 compositioning 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. 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 [30] was used to measure the energy consumption due to accessing the predictor; which is 10.9 pJ per access.

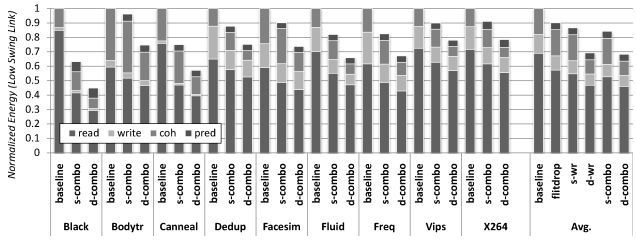
4.2 Simulation Methodology

We used M5 full system simulator to generate CMP cache block utilization traces for multi-threaded applications [31]. 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 [32]. 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.

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.



(a) full-signal swing link

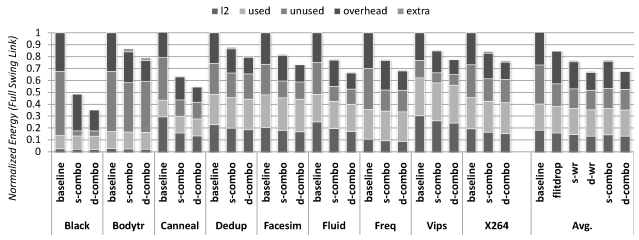


(b) low-signal swing link

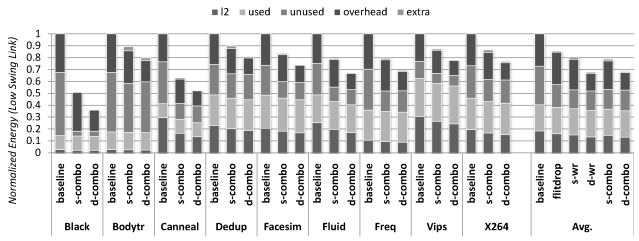
Figure 7: Dynamic energy breakdown

4.3 Benchmark Results

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.



(a) full-signal swing link



(b) low-signal swing link

Figure 8: Dynamic energy breakdown for reads

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 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 ∼69% of the total. Coherence traffic consumes the second most with ∼18% 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 ∼35% dynamic energy savings on average. 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 ∼20% and ∼16% 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 1.7% of the total energy when full-signal swing link is used, and 4.8% 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.

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 all "used" bars are equal. 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 92% 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 Write and Coherence Energy Discussion

Figure 7 shows that writebacks consume an average of 13% of total dynamic energy. Our experimental results show that when a cache block must be written back to a lower level cache or memory, on average, 34% of the words in the block are clean. The *s-combo* scheme reduces the energy consumption due to writebacks by 36% when full-swing link is used. Further savings are achieved by *d-combo*. It encodes not only body/tail flits but also head flits of the writeback packets resulting in 42% savings.

In the PARSEC benchmarks, threads running on each tile share coherent memory with other threads on other tiles. Thus, coherence protocol messages and responses are also injected to the network during the program's execution. Those protocol messages are composed primarily of single-flit packets, and contribute ∼18% of total link energy consumption. Single-flit packets, like read requests, contain ∼50% unused data. Static-encoding schemes can not reduce energy dissipation due to these packets though dynamic-encoding schemes can reduce it by up to 45.4%.

4.4 Predictor Accuracy

Figure 9 shows the distribution of prediction results with respect to various threshold values. The bars marked as *correct* show the portion of correct predictions. The bars marked as *over* show the fraction of predictions which falsely predict unused words as useful. These predictions do not cause any miss penalty but represent a lost opportunity for

energy reduction. The *miss* bars includes all predictions which falsely predict a word is not used. These mispredictions result in additional memory system packets, potentially increasing both energy and latency.

In general, as the threshold value increases, the portions of *correct* and *miss* increase while *over* decreases. This implies that the higher the threshold we choose, the lower the energy consumption but the higher the latency. Experimentally we determined that $Energy \times Delay^2$ is approximately equal across the thresholds between 1 and 12. Both latency and energy consumption become worse with threshold of 15.

We conservatively chose 1 for the threshold value in our experiments to reduce the incidence of redundant L1 misses, which may effect performance. We find the performance impact with this bias is negligible. On average, with this threshold, the additional latency of each operation is ∼0.8%. These results show that further energy savings could be achieved through improved predictor accuracy, which we leave to future work.

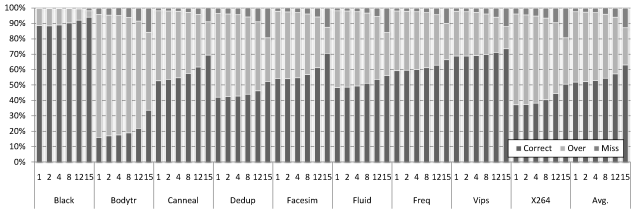


Figure 9: Breakdown of Predictions Outcomes

5. 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 20% 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 35%, on average, for both full-swing and low-swing links.

6. 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] 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.*

[10] 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.

[11] 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.

[12] 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.*

[13] 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.

[14] 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.

[15] 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.

[16] 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.*

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

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

[19] 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.

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

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

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

[23] 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.

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

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

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.