Data Criticality in Network-On-Chip Design

Joshua San Miguel
University of Toronto
Toronto, Canada
joshua.sanmiguel@mail.utoronto.ca

Natalie Enright Jerger
University of Toronto
Toronto, Canada
enright@ece.utoronto.ca

ABSTRACT

Many network-on-chip (NoC) designs focus on maximizing performance, delivering data to each core no later than needed by the application. Yet to achieve greater energy efficiency, we argue that it is just as important that data is delivered *no earlier* than needed. To address this, we explore *data criticality* in CMPs. Caches fetch data in bulk (blocks of multiple words). Depending on the application's memory access patterns, some words are needed right away (*critical*) while other data are fetched too soon (*non-critical*). On a wide range of applications, we perform a limit study of the impact of data criticality in NoC design. Criticality-oblivious designs can waste up to 37.5% energy, compared to an idealized NoC that fetches each word both no later and no earlier than needed. Furthermore, 62.3% of energy is wasted fetching data that is not used by the application. We present NoCNoC, a practical, criticality-aware NoC design that achieves up to 60.5% energy savings with no loss in performance. Our work moves towards an ideally-efficient NoC, delivering data both no later and no earlier than needed.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures—*interconnection architectures*

1. INTRODUCTION

As CMPs scale to hundreds and thousands of cores, the NoC becomes a significant factor in the cost of accessing data from memory. Traditional processor designs have focused on accessing data with minimal latency, fetching in bulk and employing prefetchers to deliver data no later than needed by the application. However, energy consumption is now a major concern, with NoC energy expecting to grow infeasibly high as we continue towards lower technology nodes [7]. Our work argues that NoCs should deliver data both no later *and no earlier* than needed to achieve ideal efficiency.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
NOCS '15, September 28 - 30, 2015, Vancouver, BC, Canada
Copyright 2015 ACM. ISBN 978-1-4503-3396-2/15/09 ...$15.00
DOI: http://dx.doi.org/10.1145/2786572.2786593

Fetching data in blocks of 16+ words is beneficial since it avoids large tag arrays, improves DRAM row buffer utilization and exploits spatial locality. But fetching some data prematurely wastes NoC energy. As an analogy, imagine multiple cars heading to the airport, each catching a different flight. Bulk-fetching implies that the cars convoy together and take the fastest route such that all cars arrive in time for the earliest flight. This is wasteful since some cars have later flights and could take a slower route that burns less fuel.

We define *data criticality* as the promptness with which an application uses data after its block is fetched from memory. When an instruction currently waiting in the pipeline uses a data word immediately upon its arrival, this word is *critical*. A *non-critical* word is fetched merely as a consequence of bulk-fetching the entire block and may (or may not) be used later by some yet to be issued instruction. Data criticality is an inherent characteristic of both an application's memory access patterns and the processor microarchitecture. In this paper, we analyze criticality and perform a limit study to estimate the amount of energy wasted in using traditional, criticality-oblivious NoC designs. Compared to a theoretical, idealized NoC that fetches each data word both no later and no earlier than needed, we find that up to 37.5% energy is wasted. In addition, we find that 62.3% of energy is wasted in fetching data words that is never used by the application at all (*dead* words). To demonstrate the significance of data criticality, we evaluate a practical NoC design that dynamically adapts to criticality. We show that a criticality-aware NoC achieves up to 60.5% energy savings with negligible impact on performance. We make the following contributions:

- We study the impact of data criticality on PARSEC [5] and SPLASH-2 [35] applications.
- We show that a criticality-oblivious NoC can waste up to 37.5% dynamic energy (17.3% on average). Furthermore, 62.3% of dynamic energy is wasted fetching dead words.
- We present NoCNoC (Non-Critical NoC), a practical, criticality-aware design that achieves energy savings of 27.3% (up to 60.5%) with no loss in performance.

2. DATA CRITICALITY

Data criticality defines how promptly an application uses a data word after its block is fetched from memory. A word is *critical* if an instruction that needs it is issued in the processor pipeline before the word arrives at the L1 cache. The instruction will stall, hurting application performance. Ideally,

```
for (i++) {
    ... = BlkSchlsEqEuroNoDiv(sptprice[i]);
}
```

(a) blackscholes.

```
for (ipar++)
for (inc++)
for (iparNeigh++) {
    ... = cell->p[ipar];

    if (borderNeigh) {
        pthread_mutex_lock();
        neigh->a[iparNeigh] -= ...;
        pthread_mutex_unlock();
    } else {
        neigh->a[iparNeigh] -= ...;
    }
}
```

(b) fluidanimate.

Figure 1: Examples of data criticality.

we would like to fetch this word through the NoC with zero time. To mask the fetching latency, modern superscalar, out-of-order processors continue executing independent instructions while waiting. However, performance still suffers as many applications have limited instruction-level parallelism (ILP). Since data is fetched in blocks, many words arrive at the cache prematurely. A word is *non-critical* if it is not needed by any instructions currently in the pipeline. Even if this word was fetched with zero latency, it would not improve performance. Words with low criticality are less sensitive to NoC latency and can tolerate longer fetch delay without harming performance.

Fig. 1 presents simplified examples of data criticality from PARSEC [5]. In blackscholes (Fig. 1a), `sptprice` is accessed sequentially, exhibiting spatial locality. However, the function `BlkSchlsEqEuroNoDiv` performs many floating point operations and takes a long time to complete. With 16-word blocks, the first iteration of the loop fetches not only `sptprice[0]` but all elements up to `sptprice[15]`. Since the execution of `BlkSchlsEqEuroNoDiv` is long, many of the elements—especially `sptprice[15]`—are fetched long before they are needed; these elements have very low criticality. Fig. 1b presents another example. Since array `cell->p` is invariant to the inner loops, there is a long delay between sequential array accesses. Synchronization can also yield non-critical words. Depending on a particle's location, `neigh->a` may or may not be enclosed by a lock. As a result, the delay between accesses may be large due to lock contention.

Data criticality is an inherent consequence of spatial locality and is exhibited by most (if not all) real-world applications. Note that compiler optimizations can reorder instructions to improve ILP and thus increase criticality, but the amount of ILP is limited by both the microarchitecture and the application characteristics. In general, low criticality can arise in the following situations:

- Long-running code between accesses.
- Interference due to thread synchronization.
- Dependences from other cache misses.
- Preemption by the operating system.

2.1 Data Liveness

We also study *data liveness* as an extreme case of criticality. A word is *live-on-arrival* (or *live*) if it is accessed at least once while in the L1 cache. With perfect spatial locality, all 16 words of a block would be live. However, in many applications, many cache words are fetched but never used prior to eviction. A word is *dead-on-arrival* (or *dead*) if it is never accessed before being evicted. Fetching dead words expends unnecessary energy.

```
for (iz++)
for (iy++)
for (ix++)
for (j++) {
    if (border(ix)) ... = cell->v[j].x;
    if (border(iy)) ... = cell->v[j].y;
    if (border(iz)) ... = cell->v[j].z;
}
```

(a) fluidanimate.

```
for (i++) {
    mNewParticles[i] = mParticles[mIndex[i]];
}
```

(b) bodytrack.

```
while (keep_going) {
    a = b;
    b = _netlist->get_random_element();
}
```

(c) canneal.

Figure 2: Examples of data liveness.

Fig. 2 presents simplified examples of data liveness. In Fig. 2a, elements of array `cell->v` are accessed sequentially by the inner loop. However, depending on the position, an element's `x`, `y` or `z` member may not be accessed. As a result, dead words may be fetched. This is common in many applications since members of data structures are stored contiguously in memory but not always needed during a given phase of the application. Dead words can also arise from irregular access patterns. In Fig. 2b, `mParticles` is not accessed sequentially but rather accessed based on `mIndex`; some elements are used more than others and some are never used at all. As a more extreme example, simulated annealing (Fig. 2c) exhibits very little spatial locality since elements of `_netlist` are accessed randomly; neighbouring elements are brought into the cache but never used.

Data liveness measures the degree of spatial locality. The more live words in a cache block, the better the spatial reuse. In general, words can be dead-on-arrival if they are evicted early or inherently unused by the application. Note that compilers can employ data packing and alignment optimizations to increase spatial locality and thus liveness, but this can potentially yield more false sharing of cache blocks. Dead words are commonly found in the following situations:

- Unused members of structs.
- Irregular or random access patterns.
- Heap fragmentation.
- Padding between data elements.
- Early evictions due to invalidations, cache pressure or poor replacement policies.

2.2 Comparison to Instruction Criticality

Instruction criticality [13, 15, 16, 31] is a measure of the relative importance of each instruction in the pipeline, typically based on how likely the instruction is to fall on the critical path of execution. For example, assume load instruction `X` accesses the word `a[0]` (i.e. the word at offset 0 in block `a`) and load instruction `Y` accesses the data word `b[0]`. Both instructions are issued, miss in the L1 cache and are now wait-

Processor	16 cores, 2 GHz, 4-wide, out-of-order, 80-instruction ROB
Cache blocks	64 B (16 words)
L1 cache	private, 4-way, 1-cycle latency, 64 kB per core
L2 cache	shared, fully distributed, 16-way, 6-cycle latency, 16 MB total
Main memory	4 banks, 160-cycle latency, 4 GB total
Cache coherence	MSI protocol
Technology node	22 nm

Table 1: CMP configuration.

Topology	4×4 mesh
Frequency	2.0 GHz
Voltage	1.14 V
Channel width	128-bit
Virtual channels	6 per port (4 flits each)
Router pipeline stages	3
Routing algorithm	X-Y dimension-order
Request/Response packet size	6B / 6+64B

Table 2: Baseline NoC.

ing for their data in the reorder buffer (ROB). NoC criticality schemes (e.g., Aergia [13]) would prioritize/deprioritize fetching either `a` or `b` to improve performance and/or save energy. This priority is based on the criticality of `X` and `Y`, deciding which of these two instructions is on the critical path of execution and more likely to stall the processor. We refer to this as *instruction criticality.* However, this form of criticality is incomplete; it does not capture data criticality. When we prioritize block `a` over `b` based only on the criticality of instructions `X` and `Y`, then we assume that all other words in the block—`a[1]`, `a[2]`, etc.—are as equally critical as `a[0]`. This may not be the case since we do not know which instructions will be issued in the future. Perhaps the application will need `a[2]` before `a[1]`. Perhaps the application does not access `a[1]` until many cycles later, wasting energy in fetching it as quickly as `a[0]`. Thus data criticality is separate from instruction criticality. Determining data criticality is challenging since it assesses words that may (or may not) be used by instructions that may (or may not) be issued soon or far into the future.

3. CRITICALITY STUDY

We characterize data criticality across a range of applications. We model a theoretical NoC design for each application—ideally tuned to its criticality characteristics—which represents the lower-bound energy consumption when fetching data both no later and no earlier than needed.[1] From this, we study the impact of criticality and estimate energy wasted in conventional, criticality-oblivious NoCs.

3.1 Experimental Methodology

We assume a 16-core CMP, configured as in Table 1. Our baseline is a conventional NoC design, configured as in Table 2. We use FeS2—a full-system x86 simulator [25]—with BookSim [18] to run applications from PARSEC [5] and SPLASH-2 [35]. Energy consumption is measured using DSENT [32]. For each application, dynamic energy is measured using the number of injected bytes, hop count, and runtime collected from FeS2 and BookSim. NoC voltage-scaling values are obtained from Sharifi et al. [29].

3.2 Measuring Criticality

To characterize criticality, we measure the fetch latency and access latency of all data words used in each application. For a data word `a[x]` (i.e., the word at offset `x` in block `a`), its fetch latency is the time elapsed from the L1 cache's request for `a` to the arrival of the data block at the cache. This is the uncore latency, much of which is contributed by the NoC. In a conventional, criticality-oblivious NoC design, the fetch latency of all words in `a` are identical. The access latency of `a[x]` is the time elapsed from the L1 cache's request for `a` to the first L1 access (hit) of `a[x]` by the application. This represents the data word's criticality; the lower the access latency (i.e., the sooner it is used), the higher the criticality.[2]

Fig. 3 shows the distributions of access latency (normalized to fetch latency) of all words accessed. For now, we ignore dead words. By normalizing to fetch latency, we effectively estimate the amount of network slowdown that can be tolerated. For example, if a word's access latency is 3× its fetch latency, then the NoC can fetch this word 3× slower (saving energy) without hurting performance. Applications are grouped based on their overall criticality from very low to very high. The distributions in Fig. 3 are cumulative. For example, in fluidanimate (Fig. 3a), 20% of all words can tolerate a network that is 10× slower than the baseline.

Words with a normalized access latency of 1× (or lower) are critical. These words are used immediately upon arriving at the L1 cache; instructions in the ROB are already waiting for them. Fig. 3 shows that non-critical words exist in all applications, even those with high overall criticality. Applications are generally written to exploit spatial locality, due to the bulk-fetching nature of conventional caches. Since processors cannot achieve perfect ILP, some words are bound to be fetched before the instructions needing them are issued. Applications with very low criticality (Fig. 3a) tend to fetch data too early. These include blackscholes and swaptions, which are financial applications with many time-consuming floating-point operations in between data accesses (Sec. 2). Fluidanimate and streamcluster also exhibit very low criticality due to heavy synchronization and long delays between accesses.

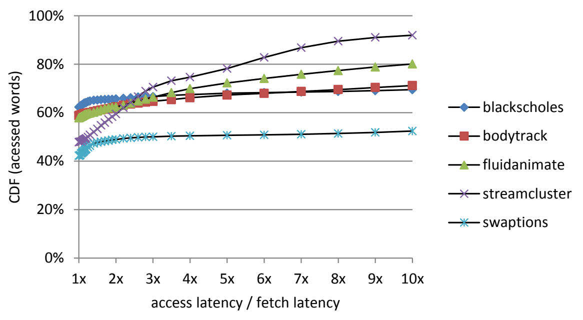
3.3 The Impact of Criticality

We aim to quantify the amount of energy wasted in conventional NoC designs, where in a given block `a`, the fetch latencies of all words—`a[0]`, `a[1]`, etc.—are equal. These designs are oblivious to the varying criticalities (access latencies) of the data words. Our goal is to model a theoretical, ideal NoC where every word is fetched such that its fetch latency is equal to its access latency; all data is delivered both no later and no earlier than needed.

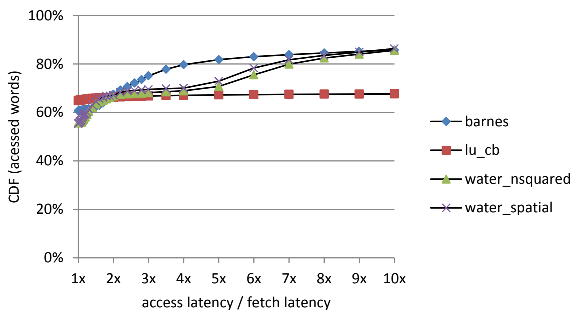
Experiments. Starting from the baseline, we divide the NoC into multiple subnetworks each operating at a different frequency and voltage. The subnetworks split the baseline 128-bit channel width. Using the distributions from Sec. 3.2, each subnetwork is assigned to fetch a subset of words that share a common access latency. The subnetwork's frequency is then configured such that the fetch latency is equal to the access latency. We perform a brute-force search of all possible subnetwork configurations to find the one that yields the

[1]Note that *just-in-time* fetching may not always be optimal since it can alter congestion patterns in the network. For this study, it simply serves as a first step towards designing criticality-aware NoCs.

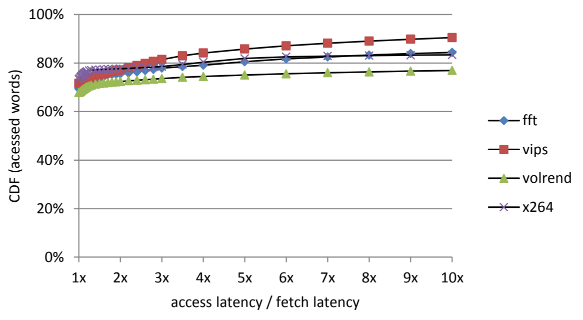
[2]Access latency encapsulates the fetch latency.



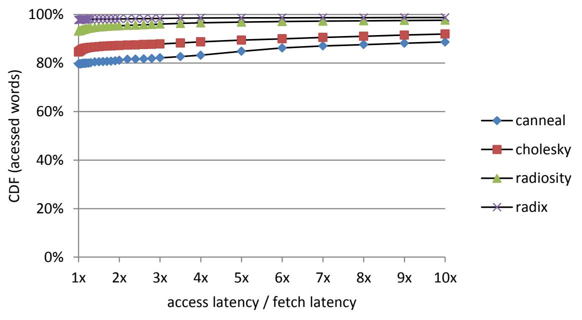
(a) Very low criticality.



(b) Low criticality.



(c) High criticality.



(d) Very high criticality.

Figure 3: Cumulative distribution functions of criticality of all words accessed. Applications grouped by degree of criticality.

subnet	access / fetch	channel	frequency	voltage
0	1×	76-bit	2.0 GHz	1.14 V
1	1× - 1.1×	4-bit	2.0 GHz	1.14 V
2	1.1× - 2×	4-bit	1.8 GHz	1.09 V
3	2× - 3.5×	4-bit	1.0 GHz	0.84 V
4	3.5× - 7×	8-bit	571.4 MHz	0.71 V
5	7× - 18×	8-bit	285.7 MHz	0.62 V
6	18× - ∞	24-bit	111.0 MHz	0.57 V

Table 3: Ideal NoC model for bodytrack.

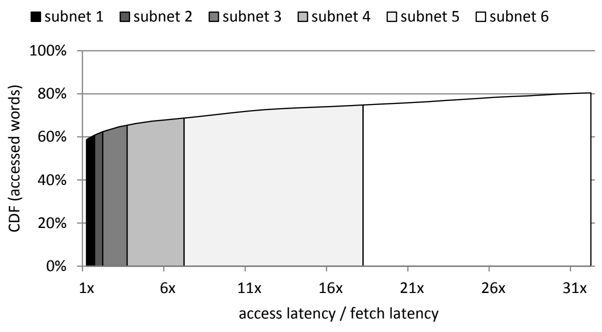


Figure 4: Word distribution of ideal NoC model for bodytrack. Areas under the curve represent subnetworks.

lowest energy. Table 3 shows the ideal NoC configuration for bodytrack.[3] This is visualized in Fig 4, where each area under the curve represents a subset of words that are assigned

[3]We do not consider channel widths narrower than 4 bits. Logic overheads of routing and allocation and sideband signals make narrow channels unrealistic.

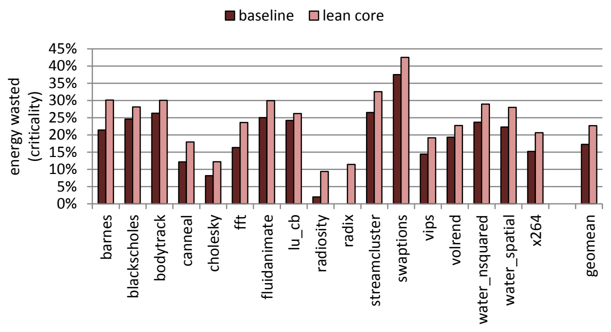


Figure 5: Energy wasted due to non-criticality.

to a given subnetwork. For example, only the words with normalized access latency between 7× and 18× are injected into subnetwork 5. Words with an access latency of 7× can tolerate a frequency of 285.7 MHz (baseline 2.0 GHz divided by 7). Each word is injected into the subnetwork with the highest fetch latency that is no greater than the word's access latency. This allows us to estimate the lower-bound energy consumed when all data is fetched both no later and no earlier than needed.

Results. Fig. 5 shows the amount of dynamic energy wasted in the baseline, criticality-oblivious NoC compared to the ideal NoC. On average, 17.3% of energy is wasted (up to 37.5% for swaptions). As expected, conventional NoCs are energy-inefficient for applications with very low criticality (Fig. 3a), expending 27.6% of unnecessary energy. Wasted energy is driven by two factors: the fraction of words that are non-critical and their degree of non-criticality (the amount

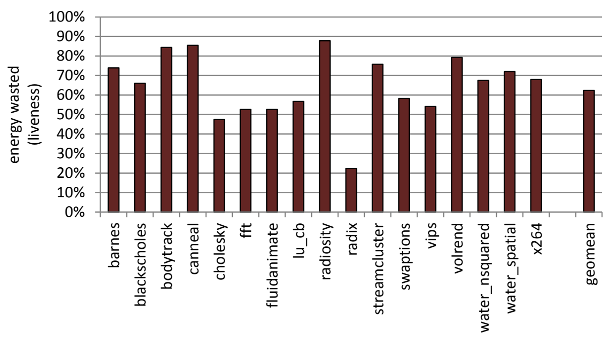


Figure 6: Energy wasted in fetching dead words.

of network slowdown they can tolerate, based on the ratio of access and fetch latency). Even though canneal and cholesky have a low fraction of non-critical words (Fig. 3d), significant energy is still wasted since they can tolerate large network slowdowns beyond 10×.

Data criticality is a function of not only the application's memory access patterns but also the processor microarchitecture. Fig. 5 also shows results for a *lean core*, which is single-issue with a 20-entry ROB. Since the lean core exploits less ILP, at any given time, there are less instructions waiting in the processor pipeline for their data. This increases the fraction of non-critical words and their access latencies, reducing overall criticality. As a result, even applications with very high criticality—such as radix and radiosity (Fig. 3d)—waste signficant energy. On average, 22.7% (up to 42.5%) is wasted with lean cores. It is important to design NoCs with criticality-awareness, particularly when running low-criticality applications or when using lean processor cores.

3.4 The Impact of Liveness

To explore the impact of data liveness, Fig. 6 shows the energy wasted in fetching dead words. On average, 62.3% (up to 87.9%) of energy is expended on fetching words into the L1 cache that are never used before being evicted. As discussed in Sec. 2, this is generally attributed to poor spatial locality and irregular access patterns. This is evident in both bodytrack and canneal, where 84.4% and 85.5% of energy is wasted. An ideally energy-efficient NoC would not consume any unnecessary energy in fetching dead words.

4. CRITICALITY-AWARE NOC DESIGN

We studied ideal, criticality-aware NoC designs in the previous section; however, in the real world, such designs are infeasible. It is impossible to achieve perfect criticality-awareness (i.e., all data is delivered both no later and no earlier than needed) because the access latency of a word is not known in advance. Thus we cannot guarantee a fetch latency that is exactly identical to the access latency. Fortunately, it is possible to implement a practical NoC design that can dynamically predict criticality with high accuracy and save significant energy. In this section, we present such a design—NoCNoC (Non-Critical NoC)—which achieves energy savings of 27.3% (up to 60.5%) with no loss in performance. A criticality-aware NoC design needs to perform three key functions:

1. Predict the criticality of a data word prior to fetching it (Sec. 4.1).
2. Physically separate the fetching of words based on their criticality (Sec. 4.2).
3. Reduce energy consumption in fetching low-criticality words (Sec. 4.3).
4. Eliminate the fetching of dead words (Sec. 4.4).

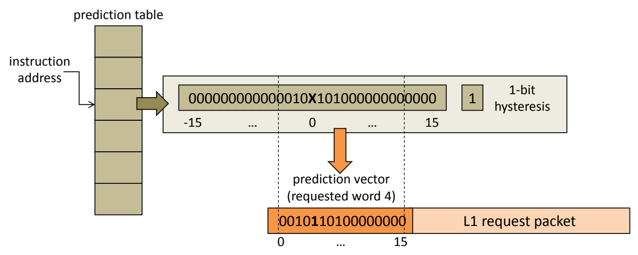


Figure 7: Data criticality predictor.

4.1 Predicting Criticality

Ideally, we want to know the exact time a word will be accessed (if at all) before fetching it. However, measuring and storing the access latencies of every word in the application is impractical. To keep overhead and complexity low, NoCNoC employs a simple binary prediction scheme: a word is predicted to be either critical or non-critical. Fig. 7 shows the hardware predictor, which is inspired from previous work [20]. The predictor is coupled with the L1 cache. It utilizes a table of 31-bit vectors indexed by the instruction address. We use untagged table entries to keep overhead low. Each vector tracks the access history of each word in the cache block based on its offset relative to the *requested word* (that caused the cache miss).

Prediction Lookup. Fig. 7 demonstrates an example prediction. A cache miss occurs requesting the word at offset 4. From the criticality vector, the words at offsets 2, 5 and 7 are predicted to be critical since they are positioned -2, +1 and +3 respectively, relative to the requested word. A 16-bit *prediction vector* is extracted and appended to the L1 request packet. Note that the requested word is always critical because there already is an instruction (which caused the cache miss) waiting to access it.

Prediction Update. Each block in the L1 cache stores the requested word offset, a pointer to the prediction table entry and a 16-bit *access vector*. Starting with the initial request, the access vector keeps track of which words in the block have been accessed by instructions. When the block arrives at the L1 cache, the contents of the access vector indicate which words are critical (since they were accessed while the block was still being fetched). The criticality prediction table entry is updated based on this access vector. A hysteresis bit is used to account for infrequent deviations from the data access pattern.

4.2 Separating Criticality

Ideally, we want each word to traverse a NoC optimized for its criticality. However, it is physically infeasible to implement such a NoC for all levels of criticality. Instead, NoCNoC employs a heterogeneous, two-network design, where one subnetwork is dedicated to critical words and the other is dedicated to non-critical words. For each L1 cache miss, the criticality predictor is invoked. The data response packet is then split into two separate packets: one containing the words predicted to be critical and the other containing those predicted non-critical. Recent work shows that using multiple physical subnetworks can improve energy-efficiency [2, 14, 34], even more so than multiple virtual networks [36]. We now have two routers per tile: one per subnetwork. With a mesh topology, this has negligible impact on chip area [4].

NoCNoC does not require any changes to the coherence protocol. Coherence still operates on a block granularity. It only requires that the processor core and L1 cache support early-restart; instructions waiting for a specific word can proceed immediately when the word arrives, without having to wait for the whole block. Entries in the L1 miss status handling registers are not cleared until after all words have arrived. All control packets—requests, invalidations, acknowledgements—are injected into the critical network. As writeback packets are not on the application's critical path, we inject them into the non-critical network.

4.3 Saving Low-Criticality Energy

Ideally, we want low-criticality words to be fetched no earlier than necessary to save energy. However, this is difficult to implement since data words have highly varying criticalities, which can change throughout the execution of the application. In NoCNoC, we employ dynamic voltage-frequency scaling (DVFS) to slow down the non-critical subnetwork. This allows us to save energy on words predicted to be non-critical, since they can tolerate a higher fetch latency. However, we must avoid lowering the frequency too much. Recall that a word is deemed non-critical if its access latency is greater than its fetch latency (i.e., it is not used until sometime after it arrives). As a result, if the frequency of the non-critical network is set too low, the fraction of words that are deemed non-critical approaches zero (since fetch latency becomes too high).

In NoCNoC, we set the frequency such that the utilization of the critical and non-critical subnetworks is balanced to prevent high congestion on either network. We define a DVFS threshold θ equal to the fraction of total NoC resources allocated to the non-critical network (e.g., if non-critical channel width is 4 bytes and critical channel width is 12 bytes, then θ is set to 25%). During runtime, in epochs of 10 ms (Sec. 4.5), we measure α, the fraction of total network traffic that is injected into the non-critical network. At the end of each epoch, if α exceeds θ, then the application is currently exhibiting low criticality, and the non-critical network is overutilized. NoCNoC responds by reducing the frequency of the non-critical network.[4] This allows us to 1) save energy in the presence of low criticality, and 2) balance the utilization of the two networks (i.e., reducing frequency increases fetch latency, which reduces the fraction of words that are deemed non-critical). If α is below θ, then criticality is high, and the critical network is overutilized; NoCNoC responds by increasing the non-critical frequency.

4.4 Eliminating Dead Words

Ideally, we only want to fetch data words that are needed by the application. However, it is impossible to determine whether or not a word will be used when fetching it. In NoCNoC, we employ speculative dead word elision to save energy. Using the same prediction scheme as with criticality (Sec. 4.1), each L1 cache is also equipped with a liveness predictor. Before fetching a block, the liveness predictor

[4]We vary frequency in steps of 250 MHz (Sec. 4.5).

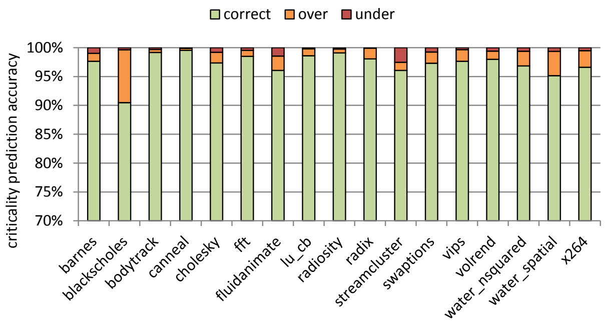


Figure 8: Criticality prediction accuracy of NoCNoC.

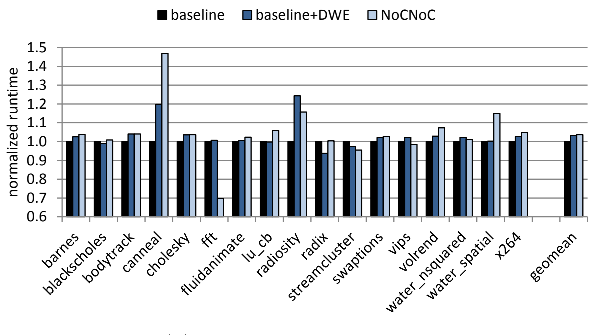
generates a 16-bit vector where each bit represents a word in the block and is set if that word is deemed to be live. This vector is appended to the L1 request packet and used to omit dead words from the data response packet. Dead word elision requires the L1 cache to use a valid bit per word instead of per block.[5] The liveness predictor is imperfect; it can occasionally mispredict a live word to be dead. When this occurs, the L1 cache must issue another request, retrieving the rest of the block.

4.5 Evaluation

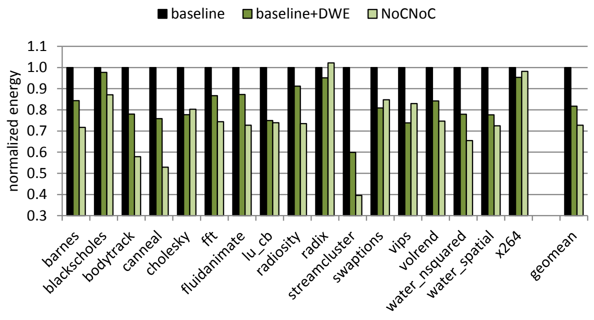
To show the promise of criticality-aware designs, we evaluate NoCNoC in terms of criticality prediction accuracy, dynamic energy consumption and application performance. We assume the same baseline CMP and NoC configurations as in Sec. 3.1. We configure NoCNoC such that the aggregate bandwidth of its two subnetworks is equal to that of the baseline conventional NoC; we implement 88-bit channels on the critical subnetwork and 40-bit channels on the non-critical subnetwork. All overheads introduced by NoCNoC—such as the 16-bit prediction vectors and the extra requests upon liveness mispredictions—are accounted for in our measurements. Predictors are initialized to assume all words are live and critical to be conservative during warm-up. We assume 4 kB prediction tables, similar to the implementation by Kim et al. [20]. Typically only 10 or fewer static load instructions cause more than 80% of L1 misses in most applications [11]. Since our prediction tables are indexed by the addresses of load-miss instructions, we expect that the table size can be reduced further while still maintaining high accuracy. Using CACTI [33], we measure the NoCNoC energy overheads (criticality and liveness predictors, and additional metadata in L1 caches) to be 2.6% of total cache energy. We employ DVFS in 10 ms epochs, assuming a 20 μs actuation overhead. θ is set to 31.25% (40-bit non-critical channels and 88-bit critical channels). DVFS metadata is piggybacked onto packets [10] and only aggregated once every 10 ms; thus overhead is negligible. In our implementation, DVFS for the non-critical network ranges from 500 MHz to 2 GHz (in steps of 250 MHz), initially set to 1.25 GHz. The critical network is fixed at 2 GHz.

Criticality Prediction Accuracy. Fig. 8 shows the criticality prediction accuracy of NoCNoC. Note that the y-axis begins at 70% for readability. A prediction is *correct* if the word is both critical and predicted-critical, or both non-critical and predicted-non-critical. On average, NoC-

[5]Coherence is still maintained on a cache block granularity.

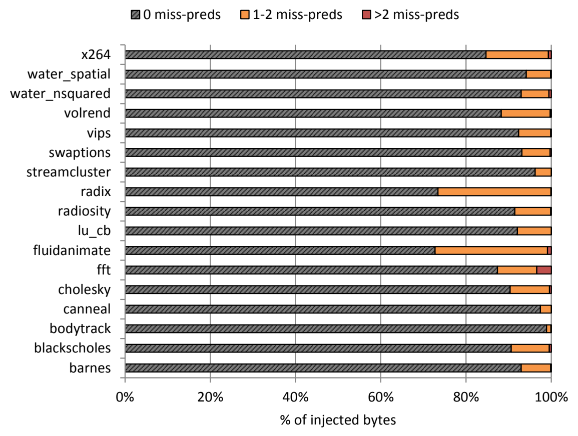


(a) Application runtime.

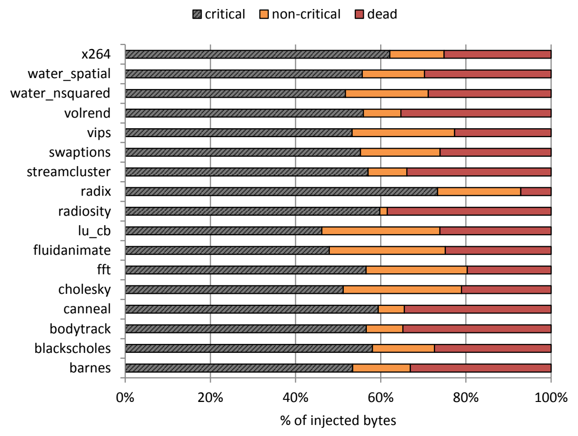


(b) Energy consumption.

Figure 9: NoCNoC performance and energy.



(a) L2 miss-predecessors.



(b) Data criticality.

Figure 10: Breakdown of injected bytes.

NoC correctly predicts the criticality of words with 97.2% accuracy. As discussed in Sec. 4.1, our predictor uses relative word offsets, which are highly effective since in most applications, spatially-colocated data elements are typically accessed in a predictable order [27]. A word is *underpredicted* if it is critical but predicted-non-critical (false negative), and *overpredicted* if it is non-critical but predicted-critical (false positive). Overpredictions increase congestion on the critical network while underpredictions stall the processor, forcing it to wait for data that has been wrongly injected into the slow non-critical network. NoCNoC achieves very low overprediction and underprediction rates of 2.2% and 0.6%.

Performance and Energy. Fig. 9a and 9b show NoCNoC performance and energy. We compare against the baseline conventional NoC with and without dead word elision (DWE). Although we demonstrate significant energy wasted on fetching dead words (Sec. 3), the inaccuracy of our liveness predictors limits the savings. On average, speculative dead word elision still achieves 18.3% energy savings compared to the baseline. NoCNoC—which includes dead word elision—sees energy savings of 27.3% on average (up to 60.5%), while increasing runtime by only 3.6%.

We note that canneal is a pathological case that performs poorly on any multi-network design. Due to its poor spatial locality (Fig. 2c), canneal exhibits the highest number of L1 misses per cycle (0.096) of any application, nearly double that of its nearest competitor (bodytrack with 0.058). This results in very heavy congestion in the NoC. By splitting the NoC into smaller subnetworks (even without reducing the frequency), the smaller channel widths increases the number of flits per packet. This increases resource contention in the NoC (buffer stalls and head-of-line blocking), which is amplified by the baseline high congestion in canneal.

Comparison to Instruction Criticality. As discussed in Sec. 2.2, instruction criticality schemes [13, 15, 16, 31] try to determine which instructions fall on the critical path of execution. Specifically, Aergia—a NoC prioritization scheme [13]—characterizes criticality based on how likely an instruction is to stall the processor pipeline. These schemes are effective in accelerating critical data blocks, but their opportunities for energy savings are limited. Unlike data criticality, they do not take into account the data that is fetched before any instructions that need it are even issued.

To illustrate this, we compare Aergia's classification of criticality (Fig. 10a) with that of NoCNoC (Fig. 10b). To identify non-critical instructions, Aergia tracks *L2-miss-predecessors*, which are older instructions that have missed in the L2 cache (suffering a longer latency to access memory). An instruction with one or two L2-miss-predecessors can tolerate some network slowdown while an instruction with >2 can tolerate even more; it is highly likely that an L2-miss-predecessor will stall the processor. Conversely, an instruction with zero L2-miss-predecessors cannot tolerate a slowdown, and thus no energy savings can be achieved. We find that instructions with zero L2-miss-predecessors make up 89.9% of injected NoC traffic, leaving only 10.1% for potential energy savings. This is due to typically low L2 miss rates and limited MLP. On the other hand, with NoCNoC, only 56.1% of traffic is critical while 43.9% can be slowed down (non-critical) or even eliminated (dead).

5. RELATED WORK

Liveness. Kim et al. exploit data liveness through a redesigned router microarchitecture [20]. Predictors have been implemented that recognize the spatial locality of memory accesses [9, 21, 30]. These focus on prefetching and cache power savings. Other techniques can identify when words and sub-blocks will no longer be used before eviction [1, 3, 19, 23, 28]. Kumar et al. exploit dead words using variable block granularity to reduce cache energy [22]. Unlike our work, these techniques do not target NoC energy.

Criticality. Exploiting instruction criticality [13, 15, 16, 31] provides less opportunities for NoC power savings compared to data criticality as shown in Sec. 4.5. Low-latency and low-power main memory modules can be used to target data criticality in the form of latency-sensitive memory accesses [8, 26]. Critical words caches dedicate a separate L2 storage array to words that were requested first in the past [17]. We observe that more than one word can be critical on each cache miss. Prioritization schemes characterize NoC packets based on latency-sensitivity [6, 12, 24]. They aim to boost performance while we aim to save energy.

6. CONCLUSION

We study the impact of data criticality in NoC design. Data criticality is an inherent consequence of spatial locality; it defines how promptly an application uses a data word after its block is fetched from memory. We find that all applications studied exhibt data criticality. Conventional, criticality-oblivious NoC designs waste up to 37.5% energy. Furthermore, 62.3% of energy is wasted fetching data that is never used by the application. To illustrate the importance of criticality-awareness, we show how NoCNoC can achieve up to 60.5% energy savings with negligible performance loss. We demonstrate that NoCs should be designed to deliver data both no later and no earlier than needed.

Acknowledgements

The authors thank the anonymous reviewers for their thorough suggestions on improving this work. The authors also thank the members of the Enright Jerger research group for their feedback. This work is supported by a Bell Graduate Scholarship, the Natural Sciences and Engineering Research Council of Canada, the Canadian Foundation for Innovation, the Ministry of Research and Innovation Early Researcher Award and the University of Toronto.

7. REFERENCES

[1] J. Abella et al. IATAC: a smart predictor to turn-off L2 cache lines. *ACM TACO*, 2005.
[2] A. K. Abousamra et al. Deja vu switching for multiplane NoCs. In *NOCS*, 2012.
[3] M. A. Z. Alves et al. Energy savings via dead sub-block prediction. In *SBAC-PAD*, 2012.
[4] J. Balfour and W. J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *ICS*, 2006.
[5] C. Bienia et al. The PARSEC benchmark suite: characterization and architectural implications. In *PACT*, 2008.
[6] E. Bolotin et al. The power of priority: NoC based distributed cache coherency. In *NOCS*, 2007.
[7] S. Y. Borkar. Future of interconnect fabric: a contrarian view. In *Proc. Int. Workshop on System Level Interconnect Prediction*, 2010.
[8] N. Chatterjee et al. Leveraging heterogeneity in DRAM main memories to accelerate critical word access. In *MICRO*, 2012.
[9] C. F. Chen et al. Accurate and complexity-effective spatial pattern prediction. In *HPCA*, 2004.
[10] X. Chen et al. In-network monitoring and control policy for DVFS of CMP networks-on-chip and last level caches. In *NOCS*, 2012.
[11] J. D. Collins et al. Speculative precomputation: long-range prefetching of delinquent loads. In *ISCA*, 2001.
[12] W. Dai et al. A priority-aware NoC to reduce squashes in thread level speculation for chip multiprocessors. In *ISPA*, 2011.
[13] R. Das et al. Aergia: exploting packet latency slack in on-chip networks. In *ISCA*, 2010.
[14] R. Das et al. Catnap: energy proportional multiple network-on-chip. In *ISCA*, 2013.
[15] B. Fields et al. Focusing processor policies via critical-path prediction. In *ISCA*, 2001.
[16] S. Ghose et al. Improving memory scheduling via processor-side load criticality information. In *ISCA*, 2013.
[17] E. J. Gieske. *Critical words cache memory: exploiting criticality within primary cache miss streams.* PhD thesis, University of Cincinnati, 2008.
[18] N. Jiang et al. A detailed and flexible cycle-accurate network-on-chip simulator. In *ISPASS*, 2013.
[19] S. Kaxiras et al. Cache decay: exploiting generational behavior to reduce cache leakage power. In *ISCA*, 2001.
[20] H. Kim et al. Reducing network-on-chip energy consumption through spatial locality speculation. In *NOCS*, 2011.
[21] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *ISCA*, 1998.
[22] S. Kumar et al. Amoeba-cache: adaptive blocks for eliminating waste in the memory hierarchy. In *MICRO*, 2012.
[23] A.-C. Lai et al. Dead-block prediction & dead-block correlating prefetchers. In *ISCA*, 2001.
[24] N. C. Nachiappan et al. Application-aware prefetch prioritization in on-chip networks. In *PACT*, 2012.
[25] N. Neelakantam et al. FeS2: a full-system execution-driven simulator for x86. poster presented at ASPLOS, 2008.
[26] S. Phadke and S. Narayanasamy. MLP aware heterogeneous memory system. In *DATE*, 2011.
[27] P. Pujara and A. Aggarwal. Cache noise prediction. *IEEE Transactions on Computers*, 2008.
[28] M. Qureshi et al. Line distillation: increasing cache capacity by filtering unused words in cache lines. In *HPCA*, 2007.
[29] A. Sharifi et al. PEPON: performance-aware hierarchical power budgeting for NoC based multicores. In *PACT*, 2012.
[30] S. Somogyi et al. Spatial memory streaming. In *ISCA*, 2006.
[31] S. T. Srinivasan and A. R. Lebeck. Load latency tolerance in dynamically scheduled processors. In *MICRO*, 1998.
[32] C. Sun et al. DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *NOCS*, 2012.
[33] S. Thoziyoor et al. CACTI 5.1. Technical Report HPL-2008-20, HP Labs, 2008.
[34] S. Volos et al. CCNoC: specializing on-chip interconnects for energy efficiency in cache-coherent servers. In *NOCS*, 2012.
[35] S. C. Woo et al. The SPLASH-2 programs: characterization and methodological considerations. In *ISCA*, 1995.
[36] Y. J. Yoon et al. Virtual channels vs. multiple physical networks: a comparative analysis. In *DAC*, 2010.