Run-Time Energy Management of Manycore Systems Through Reconfigurable Interconnects

Jie Meng Chao Chen Ayse K. Coskun Ajay Joshi
Electrical and Computer Engineering Department, Boston University, Boston, MA, USA
{jiemeng, chen9810, acoskun, joshi}@bu.edu

ABSTRACT

The active on-chip network channel width has a direct impact on the cache and memory access latency in manycore processors. A good choice of channel width improves the application performance and energy efficiency. In manycore systems, where workload patterns change significantly over time, setting network channel width statically for the average or worst-case traffic gives sub-optimal energy efficiency. This paper proposes a novel, low-cost method to reconfigure the network channel width at run time to maximize energy efficiency of applications. We analyze the effect of channel width choices for two commonly used cache hierarchies, private and distributed caches, on manycore systems with an underlying bus or crossbar architecture running parallel workloads. The proposed reconfiguration policy predicts the energy-delay product (EDP) for the currently running application at various channel widths and chooses the best fitting width to minimize EDP. The experimental results show that our policy reduces EDP by 49.3% and 23.9% for private L2 cache and 65.5% and 20.6% for distributed L2 cache on average in systems with bus and crossbar, respectively, in comparison to statically setting the channel width.

1. INTRODUCTION

Today's manycore systems already have dozens of cores on a single die (e.g., Intel's 48-core single-chip cloud [16] and Tilera's 64-core systems) and the core count is expected to increase in future. With such large core counts, the on-chip interconnect network used for L1-L2 cache and L2-memory controller communication has a significant impact on the system performance and energy. Conventional interconnect architectures for manycore systems are statically configured at design time based on the power, area and performance targets. However, such large systems undergo considerable workload changes during their lifetime, making the static configuration sub-optimal. This paper proposes a low-cost and adaptive run-time energy management approach using reconfigurable on-chip networks to improve EDP of manycore systems.

GLSVLSI'11, May 2–4, 2011, Lausanne, Switzerland.

Future manycore systems are expected to appear in a number of computing domains such as datacenters, high-performance computing clusters, and high-performance embedded systems. These domains contain a large variety of applications (scientific computing, modeling, financial applications, etc.) that can be parallelized. Similar to the applications we run today, these future parallel applications will differ in their performance characteristics, such as instructions per cycle (IPC), memory access trends, and communication intensities. Therefore, workload characteristics are expected to change dramatically at run time.

The network-on-chip (NoC) infrastructure has a direct impact on application performance and energy efficiency in manycore systems. Various design techniques improve performance and energy efficiency by adjusting the network topologies [20, 21, 14, 31], refining cache coherence protocol implementation [12, 7, 9, 26], or modifying the NoC parameters such as channel widths [27], virtual channel design [19, 29], routing algorithms[15], and flow control [22]. These techniques assume a static network architecture that does not adapt to the changes in the network traffic.

As the behavior of workloads in future manycore systems is expected to vary significantly, it is imperative to adapt the NoC architecture to the workload properties. For example, applications that are communication-intensive have higher performance when wider router-to-router channels are available. However, this performance increase results in higher NoC power consumption. Moreover, higher performance results in higher application IPC, resulting in even higher system power. This increase in system power consumption is justifiable in terms of the overall energy cost if the performance benefits are substantial and if the power budgets are not exceeded. However, for a number of benchmarks, there is limited increase in performance in such cases, causing a wastage of power if a high-performance and high-power interconnect infrastructure is selected. Recent research has started the development of reconfigurable NoC architectures to improve NoC performance and energy efficiency [28, 33, 21]. However, a unified approach considering the performance and power impact of reconfiguration on the entire system has not been developed.

In this paper, we propose a run-time NoC channel width configuration policy for optimizing the energy-performance tradeoff of manycore systems across various workloads. Our specific contributions are as follows:

- We propose a low-cost NoC reconfiguration policy for configuring the channel width in bus and crossbar topologies to minimize EDP. Our reconfiguration policy monitors application behavior through performance counters, predicts the EDP at the available widths for the manycore system based on a regression model derived offline, and selects the best-fitting width to minimize EDP. This policy is low-cost, as we reconfigure only when we predict EDP improvement.
- We provide a comprehensive analysis of the power and performance trends for a 64-core system that uses the proposed reconfiguration policy while running PARSEC [5] and NAS [3] parallel benchmarks. We explore bus and crossbar topologies, and private and distributed L2 cache architectures for the manycore system. Our policy achieves up to 89.2% and 56.6% reduction in application EDP in comparison to statically setting the network channel width for manycore systems with bus and crossbar, respectively. The average system EDP reduction across the entire set of 12 parallel applications for the bus is 65.5% for distributed L2 and 49.3% for private L2. The savings for the crossbar are 20.6% and 23.7% for distributed and private L2, respectively. These results demonstrate promising EDP savings for a highly variant workload mix running on a real-life manycore system.

The rest of the paper starts with an overview of the related work. Section 3 provides the details of the target system, the performance model, and the power model. Section 4 describes our run-time management policy for EDP optimization. Section 5 provides the experimental results, and Section 6 concludes the paper.

2. RELATED WORK

Energy management of multicore systems has traditionally focused on turning off or slowing down under-utilized resources. A number of techniques have been introduced to predict the idle time slots of cores (and other resources) to minimize the performance impact of power management (DPM) [4]. Dynamic Voltage and Frequency Scaling (DVFS) is another commonly used technique, and has been adopted in recent manycore chip design [16]. Recent research has also proposed application scheduling and dynamic global power management (e.g., [35, 17]) to improve energy efficiency. Our run-time reconfiguration technique does not utilize DPM, DVFS, or scheduling based techniques, but such techniques are complimentary and can be run in conjunction with our policy.

To address performance and power of manycore systems, run-time management techniques that explicitly account for NoC design have been proposed. Nollet et al. propose a heuristic technique for performance-aware task assignment and migration for a 3×3 system with a mesh network [30]. Similarly, a thermal management technique for 3D NoC systems performs traffic-aware downward routing and thermal-aware vertical throttling to improve throughput [8]. Fiorin et al. propose using NoC traffic information to determine resource allocation [13]. All of these techniques improve the overall system design in terms of performance and/or power but are sub-optimal for dynamically changing workload as they assume static interconnection configurations.

For reconfigurable network architectures, recent work proposes techniques such as changing the network topology [21, 28, 33], changing the virtual channel count, network fragmentation [1] or router buffer sizes [29], and packet fragmentation [18]. Dafali et al. illustrate the general process and design method of reconfigurable interconnects [11]. Such techniques focus on improving performance and/or reducing power. However, most of the prior work on reconfigurable NoCs investigate multicore systems with a small core count, use synthetic network benchmarks for evaluation, or do not perform run-time network configuration, which limits the advantages of network reconfiguration.

Our approach focuses on designing a run-time reconfiguration policy for adjusting the channel width of the interconnects. The goal is to adapt to varying workload characteristics and network traffic for improving energy efficiency of a large manycore system. We analyze parallel workloads and develop a prediction model to estimate IPC and EDP for various NoC configurations. Our run-time policy optimizes EDP by changing the channel width of a bus or crossbar only when there is a substantial performance benefit. A major distinguishing feature of our work is that we connect the network behavior with the architecture-level performance parameters to optimize the application performance and energy simultaneously.

3. METHODOLOGY

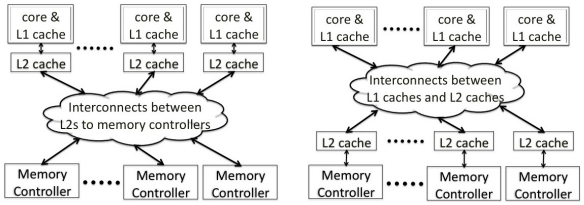
3.1 Target System

Our target system is a 64-core processor, where each core has 2-way issue and out-of-order execution. As a representative technology node for future chips [23], we assume the system is manufactured at 22 nm process and has a total die area of 400 mm^2. The cores operate at 1 GHz frequency and have a supply voltage of 0.9 V. Each core has 2 integer units, 1 floating point unit, and private 16 KB L1 instruction and data caches. The core architecture is based on the cores used in the Intel single-chip cloud [16].

We assume shared memory programming model. We explore both private and distributed L2 cache architectures and keep the total L2 size (16 MB) fixed across the two configurations. Thus, in the private L2 architecture, we have 64 L2 caches (256 KB each), while in the distributed cache architecture there are 16 L2 caches (1 MB each). The manycore system uses MESI cache coherence protocol for shared caches. Figure 1(a) and Figure 1(b) illustrate the memory hierarchy of our target system constructed with private L2 and distributed L2 caches, respectively. Our target system has 16 memory controllers, which are uniformly distributed across the chip. The architectural parameters of each core and corresponding memory hierarchy configurations are listed in Table 1.

Table 1: Core Microarchitecture Parameters

Architectural Configuration	
CPU Clock	1.0 GHz
Branch Predictor	Tournament predictor
Issue	2-way Out-of-order
Reorder Buffer	128 entries
Functional Units	2IntAlu, 1IntMult, 1FPALU, 1FPMult
Physical Regs	128 Int, 128 FP
Instruction Queue	64 entries
L1 ICache	16 KB @ 2 ns
L1 DCache	16 KB @ 2 ns
L2 Cache(s)	4-way set-associative, 64 Bytes block Private L2: 64 x 256 KB @ 5 ns Distributed L2: 16 x 1 MB @ 6 ns



(a) Private L2 (b) Distributed L2

Figure 1: Two memory architectures with caches and memory controllers distributed on the chip.

Out of the various network topologies proposed for on-chip communication in manycore systems, we explore two low-diameter network topologies: bus and crossbar, due to their ease of design and programming. Both bus and crossbar topologies use long global interconnects that are routed around the chip in a serpentine fashion. These interconnects are pipelined to reduce power dissipation and provide high throughput. Figure 2 shows the physical layout of the bus and crossbar. The arbitration block for both topologies is located in the center of the chip. For the crossbar, we adopt a multiple-write / single-read architecture; i.e., each output port is connected to multiple input ports through a dedicated channel. The channels in both topologies have flexible widths: 128-bit, 64-bit, and 32-bit. The channel widths can be configured at run time by the OS to reduce power and/or to improve performance.

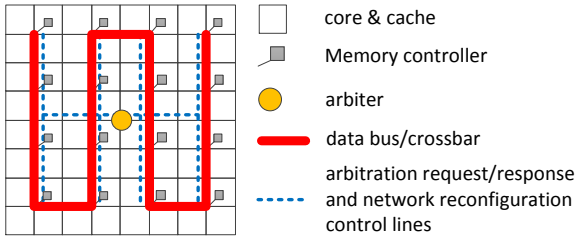


Figure 2: Layout of bus/crossbar topology. The channel width can be configured as 128, 64, or 32-bit. For the crossbar, we assume a multiple-write / single-read architecture with global arbitration.

3.2 Processor Performance Model

We use the M5 full-system simulator [6] for the performance modeling of our target system. We model our target system with the Alpha instruction set architecture (ISA) as it is the most stable ISA supported in M5. Currently, the full-system mode models a DEC Tsunami system to boot an unmodified Linux 2.6 operating system. We select parallel applications from the PARSEC benchmark suite [5] and NAS Parallel Benchmarks (NPB) suite [3], both of which have been widely used in parallel system studies.

M5 models a split-transaction bus that is configurable in both latency and bandwidth. The bus arbitration follows first-come-first-serve logic, and uses round-robin scheduling for bus accesses. Our crossbar model uses multiple-write / single-read architecture. Each dedicated channel associated with a core is modeled using bus as a building block. The access pattern follows the same arbitration logic as the bus.

We run PARSEC benchmarks in M5 with sim-large input sets and NAS with class B problem sets. For NAS benchmarks, we use a warm-up period of 1 billion instructions to get past the initialization phase and avoid cold-start effects. The start of the region of interest (ROI) (i.e. parallel phase) is pre-defined in the PARSEC hook-libraries, so we fast-forward to the ROI. We execute 1 billion instructions in the ROI with the detailed out-of-order CPUs for all the benchmarks. We collect performance statistics from M5 simulations and use them as inputs for our power model.

We use application IPC, defined in Equation (1), as the metric to evaluate the performance of the benchmarks. For single-threaded systems, IPC is a commonly used performance metric and it tends to have a strong correlation with the power consumption [10]. However, the use of IPC as a performance metric for multicore systems and parallel workloads needs special attention, considering the potential inaccuracies [2]. For example, instruction path of multithreaded workloads running on multiple processors can vary substantially, and the execution time of threads in parallel applications differ considerably. Our goal is to evaluate the impact of the run-time policy on the entire application's performance. We use the application IPC as our performance metric as it reflects the performance of the entire system. Out of execution cycles per core, the largest cycle count is used to calculate IPC as it represents the total number of cycles it takes to execute 1 billion instructions on the 64-core ($num\_cpu = 64$) system.

$$IPC_{app} = \frac{\sum_{i=1}^{num\_cpu} Committed\_Instructions_{cpu[i]}}{\max_{1 \le i \le num\_cpu} Number\_of\_Cycles_{cpu[i]}} \quad (1)$$

3.3 Processor Power Model

We use McPAT 0.7 [25] to estimate the run-time dynamic and leakage power of the cores in our target system. McPAT utilizes M5 performance results to compute the power consumption. To improve accuracy for run-time power computations, we calibrate the McPAT outputs to match the published power of Intel 48-core processor [16]. Intel system is designed using 45 nm technology, therefore we scale its power to 22 nm. The switching power dissipated by a CMOS device is proportional to $f \cdot V_{dd}^2$, where f is the operating frequency and V_{dd} is the supply voltage. While the V_{dd} dependency of the processor leakage power is exponential, we estimate it as a second order polynomial of V_{dd} around its nominal value since the V_{dd} variation is only around 20% of default setting [34]. As both our target system and the Intel chip operate at 1 GHz, we estimate the processor power of the equivalent 22 nm core using Equation (2).

$$Power_{22nm} = Power_{45nm} \cdot (\frac{V_{dd_{22nm}}}{V_{dd_{45nm}}})^2. \quad (2)$$

The reported average core power and supply voltage for Intel SCC for the 45 nm technology are 1.83 W and 1.14 V [16], respectively. For 22 nm, using Equation (2) and $V_{dd} =$ 0.9 V, we estimate the average core power as 1.14 W. We then compute the average power using McPAT across all the benchmarks and obtain the scaling factor, R, between the average McPAT power and $Power_{22nm}$. We use R to scale each benchmark's core power consumption as follows:

$$Power_{22nm\_scaled} = R \cdot Power_{22nm\_McPAT} \quad (3)$$

where $Power_{22nm\_McPAT}$ is the core power computed by McPAT and $Power_{22nm\_scaled}$ is the scaled power value we use in our experiments. A similar calibration approach has been introduced previously in [24].

We compute the L2 cache power using CACTI 5.3 [36]. As CACTI does not model the 22 nm process technology, we use CACTI to calculate the power estimation at 32 nm technology and then use the same scaling method as above.

3.4 Network Performance and Power

For the bus and crossbar network topologies, we use energy-optimized repeater-inserted pipelined channels that are 75 mm long and have 30 pipeline stages. Each pipeline stage is designed using PTM for 22 nm technology [23], is 2.5 mm long and has a latency of 500 ps. Each of the pipeline stage is optimized for low power and consumes 39 fJ of fixed energy per bit-time and 238 fJ of data dependent dynamic energy per bit-time according to SPICE computations.

We assume that the packet size is equal to a single cache line of 512 bits, and the flit size is the same as the channel width. We use $packetsize/flitsize$ to calculate the number of flits per packet. The total packet latency is calculated as a sum of the arbitration latency, time of flight for a flit, and serialization latency. The channel throughput is inverse of the latency of a single pipeline stage. Each packet is saved in its local buffer until arbitration is complete. Then, the input port serializes the packet into multiple flits and transmits them through the network.

The fixed power consumed by the network depends on physical channel width and pipelining stages, while the data dependent power depends on network packet injection rate and flit count in each packet. Across different channel widths, the total bus power of in our workloads varies from 0.6 W to 2.0 W while crossbar power varies from 25.2 W to 26.6 W. The variation in crossbar power is lower than the variation in the bus power as the fixed power component is dominant.

4. RUN-TIME RECONFIGURATION

This section discussed the proposed reconfiguration policy. The goal is to maximize the energy efficiency of manycore systems by selecting the best-fitting network channel width at run-time. We use EDP as a metric to evaluate energy efficiency. We calculate the EDP for an application as follows:

$$EDP = System\_Power \cdot application\_running\_time^2 \quad (4)$$
$$= System\_Power \cdot (\frac{max\_application\_exec\_cycs}{system\_frequency})^2$$

where $system\_power$ includes core power, cache power and NoC power. As we execute the same number of instructions in each application, we estimate the running time by dividing the maximum number of application execution cycles among all the cores by the system frequency.

The run-time reconfiguration policy is illustrated in Figure 3. First, we start running the application with 32-bit channel width for an interval of 10 million cycles (10 ms). We collect the performance statistics (L2 or memory injection rates, committed and execution instruction counts, branch misprediction rate, current IPC, etc.) during the interval and feed them into our polynomial model to estimate IPC for all the available bus (or crossbar) width configurations. In our evaluation, we collect the performance data using the M5 simulator. In a real-life system, the statistics are collected through performance counters. The performance counters in typical high-end processors today can be polled at the granularity of 10 ms (e.g., the minimal interval in Intel $VTune^{TM}$ is 10 ms). We use the IPC estimates to predict EDP for all available channel-width configurations.

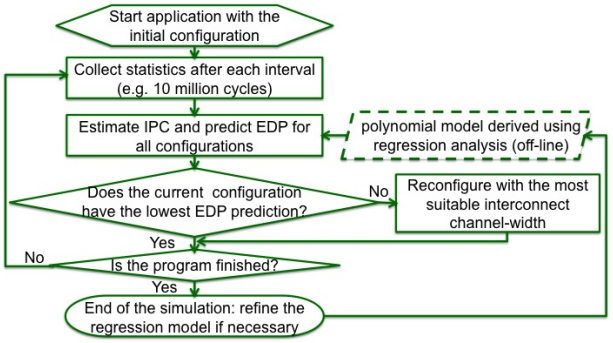


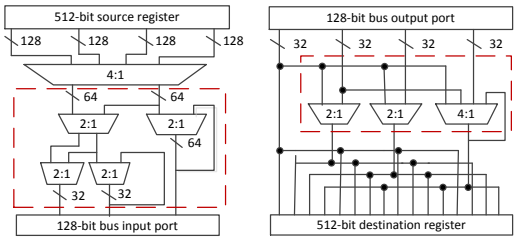
Figure 3: Flowchart illustrating the run-time reconfiguration policy with 32-bit channel-width as initial configuration.

We apply regression analysis to form a model for estimating the application IPC. Regression analysis is a technique for modeling and analyzing the relationship between a set of variables, and it is widely used for modeling and prediction. Through regression analysis, we build a polynomial offline and use it to predict IPC at run-time with the performance counter data. Since computing a polynomial at every interval has very low computational cost, the performance impact of run-time prediction is negligible. Our policy is capable of handling the performance phase changes both within an application and among applications. We predict EDP after IPC is estimated. This prediction is based on the inverse relationship between IPC and EDP, which is verified using our performance and power models. In this work, we use calibrated power models to accurately estimate system power. If fine-grained power measurements are available, such measurements could be used during offline analysis as well.

The run-time configuration decisions are made by comparing the estimated EDP values for different interconnect configurations. If the current EDP is not the lowest, we select the interconnect width with lowest EDP for the application and initiate the reconfiguration process described below. After an application finishes, we collect the power and performance statistics and use them to refine the regression model for IPC prediction. In Section 5, we provide the numerical details of our regression model.

The EDP prediction is performed at every interval, and the OS initiates the reconfiguration if necessary. If reconfiguration is required, OS throws a system call to stall all cores until all network ports are notified with the new configuration. The overall time overhead for broadcasting the new configuration is less than 100 cycles. The time for system call is in the order of 0.01ms [32], thus system call dominates the reconfiguration time overhead. As our interval length is set to 10 ms, the reconfiguration overhead is minimal.

The hardware overhead for network reconfiguration includes a set of multiplexers required to convert packets into flits of appropriate sizes. Figure 4 show hardware overhead at bus input and output ports, respectively for the 64-core system. This hardware includes 16304 2:1 multiplexers and 2048 4:1 multiplexers, and consumes less than 0.01 W at 22 nm technology. The area overhead is negligible.



(a) Bus input port (b) Bus output port

Figure 4: Hardware overhead for reconfiguration

5. EXPERIMENTAL RESULTS

In this section, we evaluate our reconfiguration policy and compare it with the bus and crossbar network with fixed channel width. We run four PARSEC benchmarks (blackscholes, bodytrack, canneal, and fluidanimate) and eight NAS benchmarks (cg, dc, ep, is, lu, mg, sp and ua).

We follow the reconfiguration policy introduced in Section 4, and first run the system for 10 million cycles and collect the performance counters for IPC prediction. In current processors, each core typically has $n = 2$ *to* 5 physical counters; however, using multiplexing we can monitor up to 2^n events. The performance data we collect are branch misprediction rate, L2 injection rate (L1 misses/cycle), memory injection rate (L2 misses/cycle), memory boundedness (L2 misses/instructions), number of load locks-store conditionals per instruction (LLSC), number of committed instructions, number of executed instructions and IPC.

We use regression with least square fit to estimate the application IPC at all available channel widths. Our regression models are constructed with the linear and cross terms of the selected performance counters. We select 10 benchmarks among the 12 applications as the training set to construct the initial regression model, and use ep and fluidanimate as the test set. The purpose of the test set is to quantify the error margin of the regression model when new workloads arrive. Recall that our policy updates the regression model with new data when needed. Figure 5 shows the regression model for estimating the IPC at channel width of 128-bit for a system with private L2 caches. We train our regression model with the initial 10 benchmarks, and the average prediction error for the training set is less than 0.001%. When an application from the training set arrives, we predict IPC accurately, select the channel width with minimum inverse IPC, and perform reconfiguration if needed. When the test benchmark fluidanimate arrives and executes for 10ms, we predict its IPC at 64 and 128-bit channels using the existing polynomial model constructed with 10 benchmarks, and the prediction error reaches 32.0%. After fluidanimate finishes, we use its performance characteristics to calibrate our regression model. We use the same approach for predicting the IPC of any new benchmark, such as ep. The prediction error for ep reduces from 29.4% to 14.8% when the model is updated with the 11^{th} benchmark.

Once we predict IPC, it is possible to estimate the EDP, as EDP is inversely proportional to IPC. Figure 6 shows the inverse of the estimated IPC for system with bus topology and private L2 from our regression model, and Figure 7 shows the EDP for each application. The figures show the same trends for all the benchmarks except fluidanimate, which is in the test set. Also, for bus topology with distributed L2 and crossbar topology, the inverse IPC has the same trend of EDP across all the benchmarks. Observing the same trends verifies that using inverse IPC provides a good estimate of the relative EDP values at 64 and 128-bit bus configurations. Thus, we use inverse IPC to make the reconfiguration decisions.

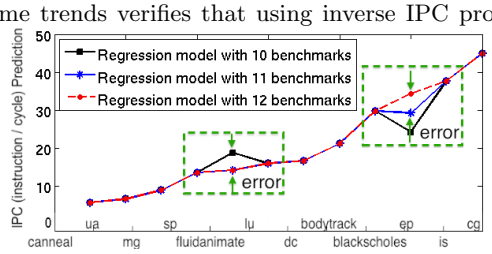


Figure 5: Regression results for predicting IPC for 128-bit bus configuration. Processor has private L2 caches.

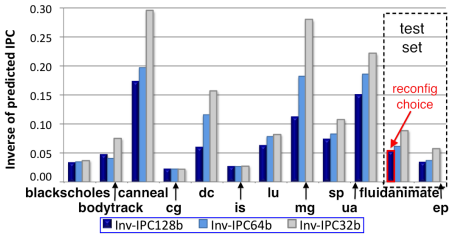


Figure 6: Inverse IPC for estimating EDP for a system with private L2 caches.

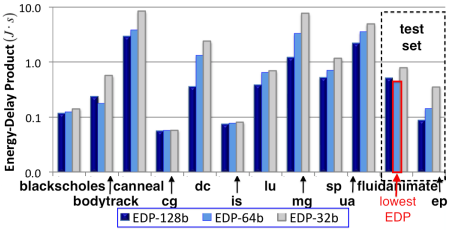


Figure 7: EDP for private L2 using different channel widths.

We see that for some applications such as cg and is, higher NoC width does not improve performance. This is due to the fact that the performance of such programs is not dominated by the network traffic, thus increasing the channel width does not provide better performance. For such cases, the run-time reconfiguration policy maintains the baseline configuration of 32-bit channel width. For other applications (canneal, dc, mg, etc.), we observe that increasing the channel width decreases EDP. In these cases, the policy reconfigures the channel width to 128-bit which provides us the best performance among the available width settings. Fluidanimate achieves highest performance when the channel width is 64-bit wide while the policy selects 128-bit due to the IPC prediction error discussed earlier. Although the misprediction causes 8% less reduction in EDP by choosing 128-bit channel width rather than 64, fluidanimate still benefits from our reconfiguration function by achieving 34.5% decrease in EDP. Overall, our policy reduces the application EDP up to 84.9%, achieving 49.3% average reduction across the entire workload set by increasing the channel width only for benchmarks with an EDP gain.

We next evaluate our policy for systems with a distributed L2 cache architecture for bus and crossbar. The simulated EDP for a system with a bus is illustrated in Figure 8. We see that for the distributed cache architecture, 128-bit channel width provides the lowest EDP for all the benchmarks. This is because the network traffic between L1 and L2 caches is heavier than the traffic between L2 caches and memory controllers in the private L2 architecture. Thus wider channel width for interconnects is preferred. Using a 128-bit bus decreases EDP by up to 89.2% and provides an average reduction of 65.5% in comparison to a 32-bit bus.

Figure 9 presents the results for a 64-core system with distributed L2 and the crossbar. For ua and fluidanimate, 32-bit channel width is a better choice in crossbar topology. This is because the proportion of crossbar power in the entire system power is much higher than the power percentage of the bus in the prior experiment. Our reconfiguration policy achieves up to 56.6% reduction in EDP in comparison to using fixed channel width of 32-bits. The average EDP decrease is 20.6% across all the benchmarks. In comparison

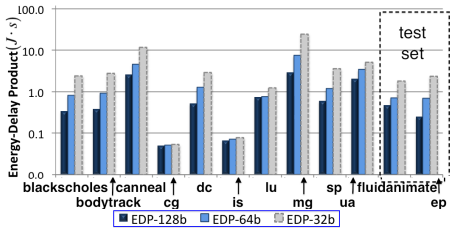


Figure 8: EDP for distributed L2 with bus topology.

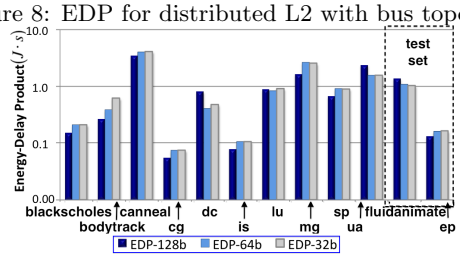


Figure 9: EDP results for a system with crossbar and with distributed L2 caches.

to statically setting the width at 128 bits, we obtain a maximum EDP decrease of 42.1% and an average EDP decrease of 9.9% by using our reconfiguration policy. The simulation results for crossbar with private L2 architecture achieve EDP savings up to 37.9% and average saving of 20.2% in comparison to statically setting the channel width to 32 bits.

Among all the experiments, we observe an application EDP decrease up to 89% when the minimum EDP setting is used instead of setting the width at 128 bits. System level EDP savings of our reconfiguration policy reach 65.5% across the 12-benchmark parallel workload set. In addition to changing channel width, it is possible to reconfigure between bus and crossbar topologies. From Figure 8 and Figure 9, we compare EDP with 128-bit crossbar and with 128-bit bus using distributed L2 architecture and observe an application EDP reduction up to 54.8% when we use crossbar instead of the bus. However, for benchmarks with low network activity (such as blackscholes), bus has lower EDP due to its lower power consumption, providing motivation to implement an approach for topology configuration.

Note that the IPC values of the applications are expected to increase if the buses are replaced with scalable NoC architectures, such as mesh or clos topologies. Also, some benchmarks benefit from running on a different ISA in terms of their performance. We plan to expand our work for a wider set of NoCs, architectures, and benchmark suites in future.

6. CONCLUSION

In this paper, we have proposed a low-cost policy to reconfigure the on-chip network channel in manycore systems to optimize the energy efficiency. The policy uses regression analysis to model the IPC of an application at the available channel widths based on performance counter data. At runtime, the policy determines the interconnect configuration with the lowest EDP for the current application, and reconfigures if necessary. We run parallel workload suites, PARSEC and NAS-NPB, for evaluation. Our technique reduces application EDP up to 89% and 57% with bus and crossbar, respectively, in comparison to statically setting the channel width. Across a workload mix of 12 parallel benchmarks, the average savings are 65.5% and 20.6%, respectively, for a manycore system with bus and crossbar with distributed L2, showing significant potential savings in real-life computing clusters running diverse sets of workloads.

7. REFERENCES

[1] S. Akram, R. Kumar, and D. Chen. Workload adaptive shared memory multicore processors with reconfigurable interconnects. In *SASP*, pages 7 –14, 2009.

[2] A. Alameldeen et al. IPC considered harmful for multiprocessor workloads. *IEEE Micro*, 26(4):8 –17, Aug. 2006.

[3] D. Bailey et al. The NAS parallel benchmarks. Technical Report RNR-94-007, March 1994.

[4] L. Benini, A. Bogliolo, and G. D. Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Trans. VLSI*, 8(3):299–316, 2000.

[5] C. Bienia et al. The PARSEC benchmark suite: characterization and architectural implications. In *PACT*, 2008.

[6] N. Binkert et al. The M5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52 –60, Aug. 2006.

[7] J. A. Brown, R. Kumar, and D. Tullsen. Proximity-aware directory-based coherence for multi-core processor architectures. In *SPAA*, pages 126–134, 2007.

[8] C.-H. Chao et al. Traffic- and thermal-aware run-time thermal management scheme for 3D NoC systems. In *NOCS*, pages 223 –230, May 2010.

[9] L. Cheng et al. Interconnect-aware coherence protocols for chip multiprocessors. In *ISCA*, pages 339 –351, 2006.

[10] A. Coskun, T. Rosing, and K. Gross. Utilizing predictors for efficient thermal management in multiprocessor SoCs. *IEEE Trans. CAD*, 28(10):1503 –1516, oct. 2009.

[11] R. Dafali et al. Key research issues for reconfigurable network-on-chip. In *ReConFig*, pages 181 –186, 2008.

[12] N. Eisley, L.-S. Peh, and L. Shang. In-network cache coherence. In *MICRO-39*, pages 321 –332, dec. 2006.

[13] L. Fiorin et al. MPSoCs run-time monitoring through Networks-on-Chip. In *DATE*, pages 558 –561, 2009.

[14] B. Grot et al. Express cube topologies for on-chip interconnects. In *HPCA*, pages 163 –174, 14-18 2009.

[15] M. Hayenga et al. SCARAB: A single cycle adaptive routing and bufferless network. In *MICRO-42*, pages 244 –254, 2009.

[16] J. Howard et al. A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS. In *ISSCC*, pages 108 –109, 2010.

[17] C. Isci et al. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO-39*, pages 347 –358, 2006.

[18] Y. Kang et al. Dynamic packet fragmentation for increased virtual channel utilization in on-chip routers. In *NOCS*, pages 250–255. IEEE Computer Society, 2009.

[19] J. Kim. Low-cost router microarchitecture for on-chip networks. In *MICRO-42*, pages 255 –266, 2009.

[20] J. Kim, J. Balfour, and W. Dally. Flattened butterfly topology for on-chip networks. In *MICRO-40*, pages 172 –182, dec. 2007.

[21] M. Kim et al. Polymorphic on-chip networks. In *ISCA*, pages 101 –112, June 2008.

[22] A. Kodi et al. iDEAL: Inter-router dual-function energy and area-efficient links for network-on-chip (NoC) architectures. In *ISCA*, pages 241 –250, 2008.

[23] K. Kuhn et al. Technology options for 22nm and beyond. In *Junction Technology (IWJT)*, pages 1–6, May 2010.

[24] R. Kumar et al. Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction. In *MICRO*, pages 81–92, Dec. 2003.

[25] S. Li et al. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO-42*, pages 469 –480, 2009.

[26] M. Marty and M. Hill. Coherence ordering for ring-based chip multiprocessors. In *MICRO-39*, pages 309–320, 2006.

[27] D. Matos et al. The need for reconfigurable routers in Networks-on-Chip. In *Reconfigurable Computing: Architectures, Tools and Applications*. 2009.

[28] M. Modarressi et al. An efficient dynamically reconfigurable on-chip network architecture. In *DAC*, 2010.

[29] C. Nicopoulos et al. ViChaR: A dynamic virtual channel regulator for network-on-chip routers. In *MICRO*, pages 333–346, 2006.

[30] V. Nollet et al. Centralized runtime resource management in a network-on-chip containing reconfigurable hardware tiles. In *DATE*, pages 234–239 Vol. 1, Mar. 2005.

[31] S. Scott, D. Abts, J. Kim, and W. Dally. The blackwidow high-radix clos network. In *ISCA*, pages 16 –28, 2006.

[32] K. Shen. Request behavior variations. In *ASPLOS*, pages 103–116, 2010.

[33] M. Stensgaard et al. ReNoC: A Network-on-Chip architecture with reconfigurable topology. In *NOCS*, pages 55–64, 2008.

[34] H. Su et al. Full chip leakage-estimation considering power supply and temperature variations. In *ISLPED*, 2003.

[35] R. Teodorescu et al. Variation-aware application scheduling and power management for chip multiprocessors. In *ISCA*, 2008.

[36] S. Thoziyoor et al. CACTI 5.1. Technical report, HP Laboratories, Palo Alto, April 2008.