Adaptive Manycore Architectures for Big Data Computing

Special Session Paper

Janardhan Rao Doppa
School of Electrical Engineering and Computer Science
Washington State University
jana@eecs.wsu.edu

Ryan Gary Kim
Department of Electrical and Computer Engineering
Carnegie Mellon University
rgkim@cmu.edu

Mihailo Isakov and Michel A. Kinsy
Department of Electrical and Computer Engineering
Boston University
mihailo@bu.edu,kinsy@bu.edu

HyoukJun Kwon and Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology
hyoukjun@gatech.edu,tushar@ece.gatech.edu

ABSTRACT

This work presents a cross-layer design of an adaptive manycore architecture to address the computational needs of emerging big data applications within the technological constraints of power and reliability. From the circuits end, we present links with reconfigurable repeaters that allow single-cycle traversals across multiple hops, creating fast single-cycle paths on demand. At the microarchitecture end, we present a router with bi-directional links, unified virtual channel (VC) structure, and the ability to perform self-monitoring and self-configuration around faults. We present our vision for *self-aware manycore architectures* and argue that machine learning techniques are very appropriate to efficiently control various configurable on-chip resources in order to realize this vision. We provide concrete learning algorithms for core and NoC reconfiguration; and dynamic power management to improve the performance, energy-efficiency, and reliability over static designs to meet the demands of big data computing. We also discuss future challenges to push the state-of-the-art on fully adaptive manycore architectures.

CCS CONCEPTS

• **Computer systems organization → Interconnection architectures**; *Fault-tolerant network topologies*; • **Hardware** → *Power and energy*; • **Computing methodologies** → Machine learning;

KEYWORDS

Adaptive manycore architectures, Big data computing, Interconnect networks, Power management, Machine learning.

ACM Reference format:
Janardhan Rao Doppa, Ryan Gary Kim, Mihailo Isakov and Michel A. Kinsy, and HyoukJun Kwon and Tushar Krishna. 2017. Adaptive Manycore Architectures for Big Data Computing. In *Proceedings of NOCS '17, Seoul, Republic of Korea, October 19–20, 2017,* 8 pages.
https://doi.org/10.1145/3130218.3130236

1 INTRODUCTION

Computing systems design is at an inflection point today. Emerging big data applications such as machine learning, graph processing, image processing, databases, etc. are often massively multi-threaded with extremely high computation demands. This has led to the emergence of many-core architectures with hundreds to thousands of cores [1, 4, 7, 9]. On the technology end, however, chips today are highly power-constrained (due to the end of Dennard voltage scaling) and face reliability and process variation challenges (due to sub-nm technology nodes).

To address the performance, energy-efficiency, and reliability needs of big data computing systems, we envision *self-aware manycore architectures* that automatically adapt their behavior to accommodate the dynamic needs of applications/users, performance and energy constraints, and resource availability. This adaptivity can span all the way from the application (e.g., task mapping/scheduling) to the core (e.g., DVFS/power-gating) to the interconnect fabric (e.g., DVFS/routing). We argue that machine learning techniques can be leveraged to reason about and manage the on-chip resources to achieve the desired performance, energy, and reliability trade-offs.

To achieve this vision, we need (a) a highly adaptive and configurable microarchitecture substrate that exposes control knobs to system software, and (b) machine learning algorithms that enable the system software to learn how to vary these control knobs to achieve system-level goals. Together, this can enable manycore systems to adapt to conditions seen during run-time (e.g., application characteristics, process-variations, aging components), while accommodating dynamic user constraints.

This paper brings together three bodies of work spanning circuits, microarchitecture, and machine learning algorithms to realize the vision of adaptive manycore architectures:

- First, in Section 2, we present the microarchitecture of a highly configurable router called RAIN that provides fine-grained control of its resources (VC buffers and links) for efficiency, while providing protection against faults with mechanisms for self-monitoring and self-configuring to create deadlock-free routes around unreliable components of the chip.
- Second, in Section 3, we present novel link circuits called SMART that provide fine-grained control of link repeaters, to enable the creation of single-cycle long-range on-chip links

on demand. This helps reduce latency. Running SMART links at lower frequencies also helps reduce energy.

- Finally, in Section 4, we present our vision for self-aware manycore architectures and argue that machine learning techniques are very appropriate to efficiently control various configurable on-chip resources to realize this vision. We provide concrete algorithms for NoC reconfiguration and dynamic power management to improve the performance and energy-efficiency over static designs.

We demonstrate the benefits of these adaptive schemes over baseline static schemes, making a case for adaptive self-aware manycore architectures. We also discuss exciting open research problems in this space in Section 5.

2 CONFIGURABLE ROUTER MICRO ARCHITECTURE FOR ADAPTIVE ROUTING

Adaptive manycore systems require on-chip networks that can dynamically reconfigure themselves to application traffic needs, e.g., high-bandwidth and user-specified constraints, e.g., hard real-time. Furthermore, these NoCs should make every effort to avoid compute resource from becoming disconnected, or deadlocks, due to defective routers and faulty links.

To this end, we propose RAIN (*Resilient Adaptive Intelligent Network-on-Chip*). RAIN makes conventional NoC routers highly configurable and resilient by augmenting conventional wormhole routers with four additional features: (1) bidirectional physical links, (2) a common pool of virtual channels (VCs), (3) a routing cost table that keeps track of latencies associated with previous routing decisions, and (4) an intelligence unit which contains a monitoring module (Mo) and a reconfiguration module (Re). Figure 1 shows the RAIN router architecture. RAIN is able to accommodate user-specified constraints and coordination across traffic paths. We also develop algorithms for fault resiliency at the on-chip network level through *self-monitoring* and *self-configuration.*

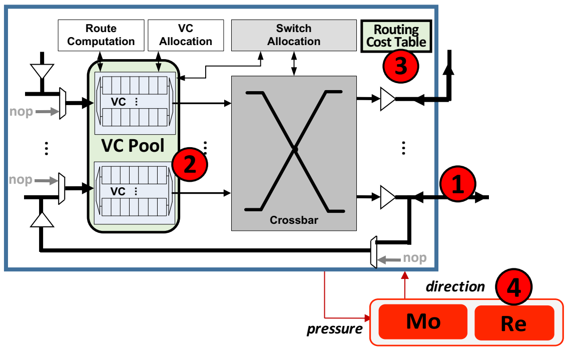


Figure 1: Resilient Adaptive Intelligent Network-on-Chip Micro-Architecture

2.1 RAIN Architecture Design Approach

Bidirectional physical links for efficient high-performance

In [16], Cho et al. introduced a bandwidth-adaptive network where the link bisection bandwidth can adapt to changing network conditions using local state information. The general design approach for bandwidth-adaptive networks is to merge unidirectional links between network node pairs into a set of bidirectional links. Each new bidirectional link can be configured to deliver packets in either direction. The links can be driven from any one of the nodes connected to it. There are local arbitration logic and tristate buffers to ensure that two nodes do not simultaneously drive the same wire. Figure 2 illustrates how the egress buffers' occupancy rate can be used to locally arbitrate the link bandwidths. First, there is sensing, followed by a reconfiguration step.

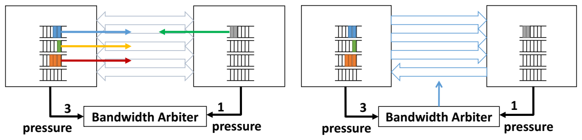


Figure 2: Local bandwidth arbitration using egress buffer pressures

The main problem with local decision making in a bidirectional link router system is the susceptibility to *head-of-line-blocking* effects. Figure 3 depicts a 2D-mesh network case using XY dimensional order routing on four flows (A, B, C, and D). Based on the pressure between nodes (1, 0) and (1, 1), a local decision is made to allocate three links from (1, 0) to (1, 1). Yet, due to the sharing of the bandwidth resources among flows A, C and D between (1, 2) and (1, 3), only one third of the allocated bandwidth between (1, 0) and (1, 1) can reach *Destination*$_A$. This local greedy approach can lead to the throttling or starvation of other flows, in this case flow B.

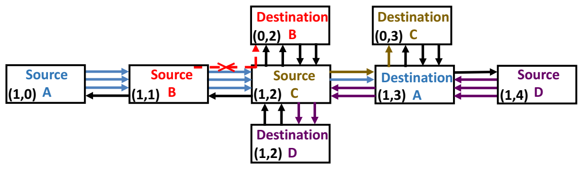


Figure 3: Illustration of head-of-line effect due to local decision making.

The RAIN architecture counters the *head-of-line-blocking* effects by allowing the coordination of direction changes and the collective arbitration of multiple links. Figure 4 illustrates the coordinated scheme. The regional information is used to biased the local decision to enable complementary effects of the distributed bandwidth adaptation. It is worth noting that this part of the approach does not require a separate network. The inter-arbiter communication logic consists of an additional three bits and can be bundled with the credit wires of the original bidirectional link architecture. The RAIN design uses two networks for resiliency.

Unified Virtual Channel Structure

Instead of having a set of virtual channels strictly associated to a given port as seen in the conventional router, the RAIN router has a pool of virtual channels that can be shared among the ports. It follows the unified virtual channel structure approach of the ViChaR [5] router architecture. With this approach, virtual channels are not statically partitioned and fixed to input ports, rather they are communal resources dynamically managed by the reconfiguration module. This approach prevents a faulty buffer from impacting any particular port or rendering a port unusable. Furthermore,

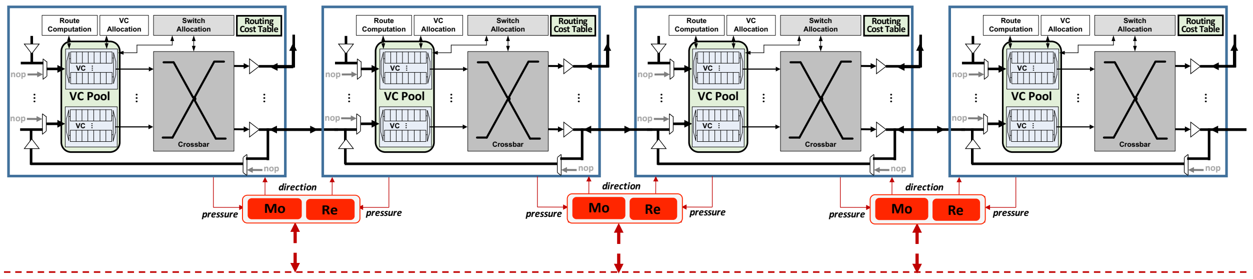
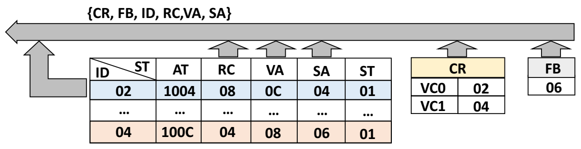


Figure 4: Autonomous globally state aware polymorphism

the unified VC design lends itself extremely well to the use of the bidirectional links in the architecture. It allows for the link direction switching to be coupled with buffer space reallocation. With more available VCs to select from, link direction switching is more efficient.

Self-monitoring and Self-reconfiguration

The RAIN router monitors its: (1) routing costs of packets, (2) buffer allocation and utilization, and (3) link and buffer operating states. First, there is a learning phase where the router collects network state information, then learns and forecasts communication patterns. Second, there is a monitoring phase to validate the information learned in the first phase. In the final phase, routing tables are updated. The collection of network state information uses an augmented credit message format. Besides the conventional credit information (CR), free buffer spaces (FB), header flit arrival time (AT), route computation latency (RC), virtual channel allocation latency (VA), and switch allocation latency (SA) information on downstream routers are sent back [15]. Figure 5 shows the information pieces added to the credit message. Instead of sharing the network link bandwidth with program data, a secondary bufferless network is created to route the augmented credit messages [29]. This credit network sends two types of messages: one contains the credit information when the header and the body flits are passing through the router and the second has the routing state information when the tail flit passes through the router. Tables shown in Figure 5 are stored in the *Routing Cost Table.* The secondary non-interfering bufferless network adds extra resiliency to the router. It guarantees that the network is monitored and state information are collected when in the presence of main network failures. A router can identify a faulty buffer, link, or router by examining FB, AT, RC, VA, and SA collected data against expected values.



ID \ ST	AT	RC	VA	SA	ST
02	1004	08	0C	04	01
...	...	...	...	...	...
04	100C	04	08	06	01

CR	
VC0	02
VC1	04

FB
06

Figure 5: Augmented credit message.

The reconfiguration is done in three places: the routing cost table, physical link direction, and virtual channel association to physical links. To ensure a deadlock-free network reconfiguration, a topology checker algorithm is run in the intelligence unit (cf., Figure 1 (4)) to determine if a change in link direction or node availability will affect a *cut-element.* a *cut-element* is an element whose removal breaks network connectivity and an element may be a vertex or an edge. The intelligence unit builds a network connectivity map and marks all cut-elements using a fully distributed *depth-first search* algorithm [18, 19]. This approach assumes that the on-chip network has been initially converted to a channel dependency graph [13] even with link direction changes.

2.2 Experimental Results

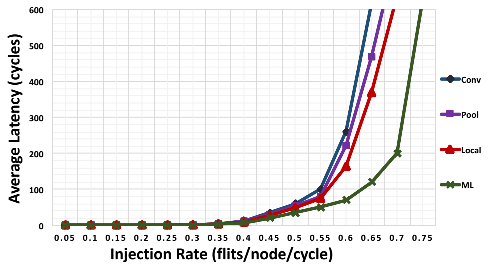


Figure 6: Transpose benchmark saturation results.

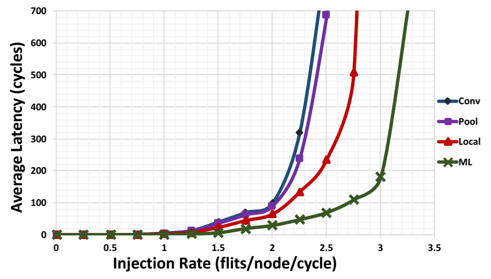


Figure 7: Synthetic Aperture Radar (SAR) saturation results.

To test the efficiency of our router design, flows from both synthetic benchmarks and real applications (e.g, Shuffle and SAR Image formation) are used. Synthetic Aperture Radar (SAR) is a radar technique for emulating the effects of a large-aperture physical radar, whose construction is not feasible, with a smaller aperture (antenna) radar. The 2-D FFT image formation algorithm in SAR is computationally very demanding and generally classified as a high-performance computing application. The application was profiled and the inter-module communication was simulated. Injection rate is correlated to image size. The network is an 8×8 2D-Mesh. The router has a pool of 32 virtual channels and 4 slots per virtual channel. The Heracles [14] RTL simulator is used for all the experiments. Heracles' injector cores are used to create network traffic. Figures

6 and 7 show the Transpose and the SAR benchmarks latency results, respectively. For the conventional router (**Conv**), the unified virtual channel design (**Pool**), and the unified virtual channel plus bidirectional physical links with local decision making (**Local**), the *Adaptive Dimensional Order Routing* (AD-DOR) is used for routing. For the globally-aware design, the neural network based predictive routing [15] is used (**ML**).

3 CONFIGURABLE LINKS CIRCUITS FOR ADAPTIVE CONNECTIONS

The fundamental equation for the latency of a packet in a NoC is as follows [17]

$$T_P = H \cdot (t_r + t_w) + T_s + \sum_{h=1}^{H} t_c(h) \qquad (1)$$

It has a fixed component for router (t_r) + link (t_w) delay, which gets multiplied by the number of hops H; a constant serialization delay T_s for multi-flit packets equal to the number of flits minus one, i.e., ($\lceil L/b \rceil - 1$), where L is the packet length and b is the link bandwidth; and a variable delay depending on contention at every hop ($t_c(h)$).

A decade of research in NoCs [3, 10, 23] coupled with technology scaling, has enabled microarchitectures with single-cycle routers (i.e., t_r=1) and single-cycle links connecting adjacent routers (i.e., t_w=1). This is the state-of-the-art today. However, network latency still goes up linearly with H. As core counts increase, H inevitably increases (linearly with k in a $k \times k$ mesh). As we add hundreds to thousands of cores on a chip [1, 4, 7, 9] for the big-data era, high hop counts will lead to horrendous on-chip network traversal latency and energy creating a stumbling block to core count scaling.

We propose to design NoCs with adaptive link circuits, that can reconfigure to create single-cycle connections between any two cores. In this section, we first present an analysis of multi-mm link circuits to demonstrate why this is feasible from a circuit point of view. Next, we present the micro-architecture of our configurable interconnect fabric called SMART [25] that enables single-cycle traversals across multiple-hops. We then present a flow control scheme to setup SMART paths dynamically. Finally, we demonstrate how SMART can be leveraged to perform efficient on-chip power management.

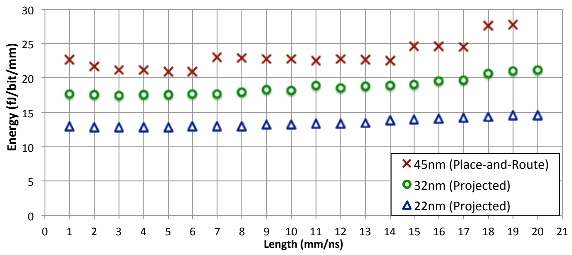
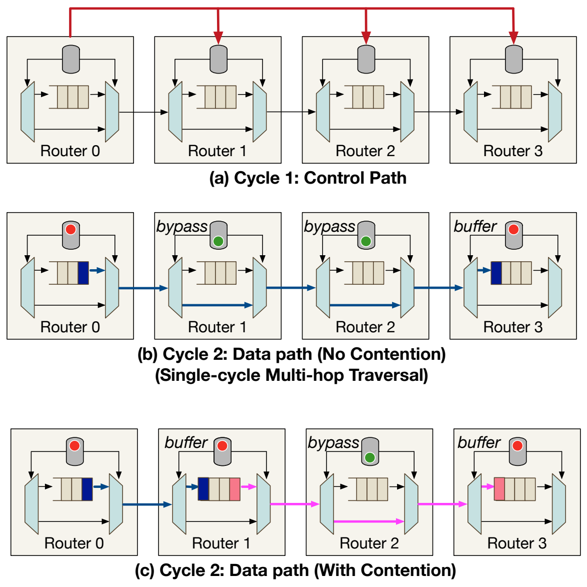


Figure 8: Delay of Repeated Wires. Repeater Spacing = 1mm, Wire Spacing ~3.DRC$_{min}$

3.1 Single-Cycle Multi-mm On-Chip Links

On-chip wires are laid out as multi-bit buses between a driver (single/multi-stage inverter) and a receiver (clocked latch/flip-flop). Wire delay depends on the effective resistance (R) times capacitance (C) values. Both R and C go up linearly with the wire's length. The C includes capacitance to ground, and capacitance between wires (i.e., coupling capacitance).

We did a design-space exploration on wire-delay using a commercial 45nm technology node and commercial CAD tools to see how fast on-chip wires are. We observed that wires can enable 13 mm signaling at a GHz by adding repeaters (inverters or buffers) at regular intervals and increasing wire spacing to (to reduce coupling capacitance), as shown in Figure 8. This length can be increased further if custom repeaters were to be designed [6] or circuit techniques like crossover and shielding used to lower crosstalk [30]. Moreover, though wires are not becoming any faster, since on-chip clock frequencies have plateaued, and chip sizes remain fairly constant due to yields, we can conclude that wires are fast enough to provide single-cycle communication between any two cores in today's and future technologies.



Router 0
Router 1
Router 2
Router 3
(a) Cycle 1: Control Path
bypass
bypass
buffer
(b) Cycle 2: Data path (No Contention) (Single-cycle Multi-hop Traversal)
buffer
bypass
buffer
(c) Cycle 2: Data path (With Contention)

Figure 9: SMART Control Path and Datapath.

3.2 Datapath: Reconfigurable Repeaters

Though on-chip wires are fast enough for single-cycle communication, laying out dedicated all-to-all wires on-chip is infeasible due to area and power constraints. Instead, many-core architectures use a NoC with short-distance links, each controlled by a router. We propose to create single-cycle long-distance wires by connecting multiple short-distance links with reconfigurable repeaters. Each repeater either operates in a *buffer mode*, latching the incoming signal like a conventional clocked receiver, or in a *bypass mode*, forwarding it to the next repeater without latching like a conventional repeater. Thus we can send signals multiple-mm on-chip by setting intermediate repeaters to act in bypass modes, and the destination to act in a buffer mode. The micro-architecture of our proposed single-cycle multi-hop datapath is shown in Figure 9(b). We call this SMART (Single-cycle Multi-hop Asynchronous Repeated Traversal) [6, 25]. The reconfigurable repeaters are embedded within each router.

3.3 Control Path: Single-cycle Reconfiguration

SMART paths can be setup in myriad ways. If the application's communication pattern is completely known in advance, SMART paths can be circuit-switched and created by configuring the repeaters right before running the application [6]. If the application's communication pattern is dynamic, SMART paths can be setup over additional control wires, which are shown in Figure 9(a). In the first cycle, the winner of the switch at Router R0 requests a SMART path of length 3 over its control wires. If there is no contention, it gets the full-path, and can perform a 3-hop traversal in the next cycle, as shown in Figure 9(b) for the blue flit. In case of contention, however, each router prioritizes its own local flits over bypassing flits, and sets the repeater to buffer mode, and sends its own flit out instead. This is shown in Figure 9(c) where the blue flit has to stop, and the pink flit uses the link between Routers 1 and 2 in Cycle 2.

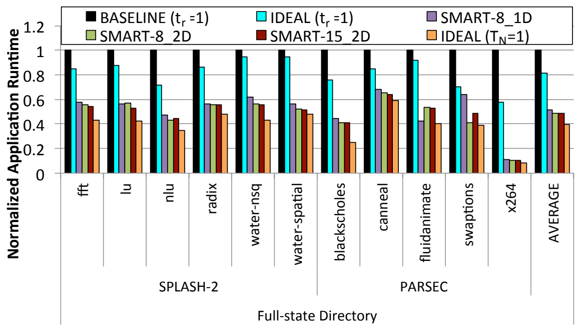


Figure 10: Performance of SMART with a Shared L2 Design.

SMART paths are thus opportunistic. This is because the underlying datapath does not have all-to-all connections. Thus flits may get partial bypass paths in case of contention. However, most modern applications do not have heavy traffic over the NoC, as most memory requests get filtered by L1 caches. Figure 10 thus shows that SMART provides 49-52% latency reduction on average with PARSEC applications over a shared distributed L2 design (which is highly sensitive to NoC latency).

In summary, SMART optimizes network latency as follows:

$$T_P = \lceil (H/HPC) \rceil \cdot (t_r + t_w) + T_s + \sum_{h=1}^{H} t_c(h) \qquad (2)$$

where *HPC* stands for number of Hops Per Cycle, and depends on contention. We reduce the effective number of hops to $\lceil (H/HPC) \rceil$. The maximum value of HPC, is known as HPC_{max} and depends on the wire delay, tile size, and clock frequency.

3.4 SMART with DVFS

As discussed above, the maximum distance (in hops) that SMART can achieve is HPC_{max}. SMART can provide an additional runtime knob for reducing energy without any loss of performance - reduce the clock frequency. In conventional DVFS, this helps lower energy, but at the cost of increased latency. In SMART, however, a lower clock frequency can help increase HPC_{max} which can reduce latency in cycles, countering the overall increase in clock period [28]. Figure 11(a)-(c) illustrates this idea. There would be an optimal frequency that can provide an overall reduction in EDP. We sweep the design-space with different values of link frequency and plot the overall NoC delay vs. energy in Figure 11(d). The number next to each configuration represents the frequency multiplier. Traditional DVFS (MESH-F2) lowers energy but increases delay. Uniform frequency scaling associated with router voltage scaling (SMART-R2L2 and SMART-R4L4) improves energy, however increases delay. Running the router (R) at a high-frequency, and links (L) at a lower frequency (SMART-R1L2 and SMARTR1L4), enable 14% reduction in latency and 52% reduction in energy. We can leverage application level behavior, as we discuss later in Section 4, to enhance the DVFS policy further.

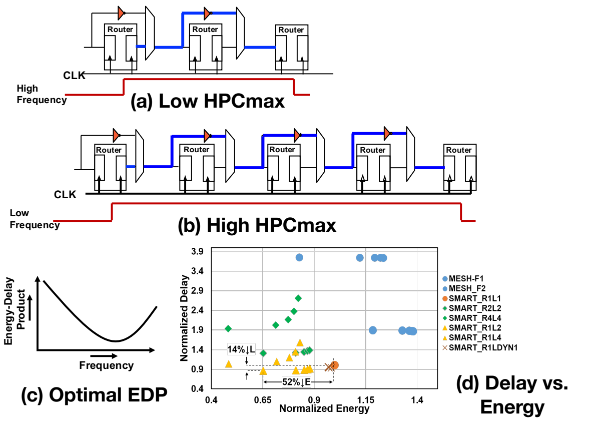


Figure 11: SMART NoC with DVFS - Adaptive HPC_{max}.

4 MACHINE LEARNING FOR ADAPTIVE CONTROL

In this section, we first describe our vision for self-aware manycore architectures and associated challenges. Subsequently, we provide some candidate machine learning solutions to realize this vision and a concrete instantiation for dynamic power management to illustrate the main ideas.

4.1 Self-Aware Manycore Architectures Vision

In today's manycore systems, the behavior of the system relies on many control knobs that control different aspects of the processor. Through careful manipulation of these knobs, we can dynamically adapt different components of the manycore system depending on the situation at hand. We envision that manycore computing systems should automatically adapt their behavior to accommodate the dynamic needs of applications/users, performance and energy constraints, and resource availability. To achieve this goal, we need online learning algorithms that enable the system to learn how to vary these control knobs so that the system can adapt to conditions seen during run-time (e.g., application characteristics, process-variations, and aging) while accommodating dynamic user constraints.

Core Adaptation

We can dynamically adapt the voltage and frequency associated with each core using learned power management policies to optimize the energy consumption subject to performance constraints. Similarly, we can reconfigure the cores/accelerators depending on the workload to improve the performance.

Interconnect Adaptation

We can dynamically adapt the voltage and frequency of network elements to optimize the energy consumption subject to performance constraints. Similarly, we can reconfigure the interconnection network on-the-fly to improve the performance and energy-efficiency of the system.

Application Adaptation

Task mapping and task scheduling will directly impact the performance and energy consumption of the system. Therefore, it is important to learn policies that can score different types of cores and accelerators based on their suitability for a given task. Additionally, task scheduling policies should optimally consider and use all available resources.

4.2 Candidate Machine Learning Solutions

Core and NoC Reconfiguration

The space of physically feasible core and NoC reconfiguration designs is combinatorial in nature. Our goal is to find the design that minimizes a given cost function O. Machine-learning techniques can enable the problem-solver (a computational search procedure) to make intelligent search decisions to achieve computational efficiency for finding (near-) optimal solutions over non-learning based algorithms [8]. The STAGE algorithm [2] is very appropriate to solve this problem. STAGE learns an evaluation function based on the data of already explored designs, which is used to guide the search towards high-quality designs. The main advantage of STAGE over popular algorithms such as simulated annealing (SA) and Integer Linear Programming is that it tries to learn the structure of the solution space, and uses this information in a clever way to improve both convergence time and the quality of the solution. As the system size increases, this aspect of STAGE is very advantageous to (1) improve the design-validate cycle before mass manufacturing; and (2) dynamically adapt the designs for new application workloads.

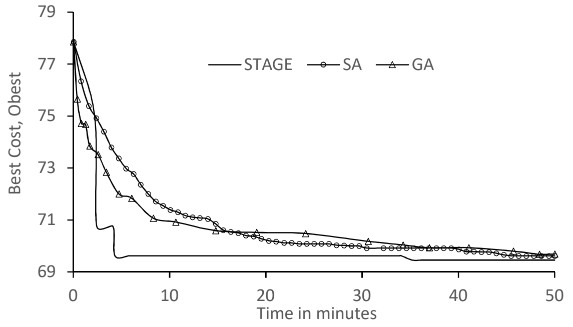


Figure 12: Performance comparison among STAGE, SA, and GA for 3D NoC design optimization.

In a recent work, we undertook a comparative performance analysis between STAGE, SA, and genetic algorithm (GA) to design a TSV-enabled 3D NoC architecture [22]. Fig. 12 shows the communication cost of the optimized network from the STAGE, SA, and GA algorithms as a function of time. We can see that STAGE uncovers high-quality designs very fast (within 5 minutes). On the other hand, SA and GA reach O_{best} more gradually compared to STAGE, and even after 50 minutes, their respective O_{best} does not reach the same solution as STAGE. We conjecture that with the increase in the design space due to large system sizes and emerging technologies (e.g., Monolithic 3D integration), STAGE will be even more efficient than SA and GA.

Adaptive Control

Reinforcement Learning (RL) and Imitation Learning (IL) are two popular machine learning approaches for learning control policies [27]. IL is considered to be an exponentially better framework than RL for learning sequential decision-making policies, but assumes the availability of a good Oracle (or expert) policy to drive the learning process. At a very high-level, the difference between IL and RL is the same as the difference between supervised learning and exploratory learning. In the supervised setting, the learner is provided with the best action for a given state. In the exploratory setting, the learner only receives weak supervision in the form of immediate costs and needs to explore different actions at each state, observe the corresponding costs, and learn from past experiences to figure out the best action for a given state. From a complexity perspective, when it is possible to learn a good approximation of the expert, the amount of data and time required to learn an expert policy is polynomial (quadratic or less) in the time horizon (i.e., number of decision steps). However, near-optimal RL is intractable for large state spaces. For large system sizes where the state space grows exponentially with the number of cores, RL methods may not scale well.

To efficiently create control policies offline for different application workloads, IL is a better choice if we can construct a good oracle policy. We provide a concrete IL methodology for dynamic power management and show its effectiveness [21]. RL formulations can be employed for online learning, but we advocate the use of more recent algorithms that take a policy search view instead of Q-learning [27].

4.3 Dynamic Power Management: A Case Study

Problem Description

The design of high-performance manycore chips is dominated by power and thermal constraints. Voltage-Frequency Islands (VFI) has emerged as an efficient and scalable power management strategy [26]. In such designs, effective VFI clustering techniques allow cores and network elements (routers and links) that behave similarly to share the same Voltage/Frequency (V/F) values without significant performance penalties. Naturally, with time-varying workloads, we can dynamically fine-tune the V/F levels of VFIs to further reduce the energy dissipation with minimal performance degradation. For applications with highly varying workloads, machine learning (ML) methods are suitable to fine-tune the V/F levels within VFIs.

Optimization Objective

Consider a manycore system with n cores. Without loss of generality, let us assume that there exist k VFIs. The dynamic VFI (DVFI) control policy π, at each control epoch t (where N is the total number of epochs), takes the current system state s_t and generates the V/F allocation for all k VFIs:

$$\pi : s_t \longrightarrow \{V_1/F_1, V_2/F_2, \cdots, V_k/F_k\} \tag{3}$$

Given a VFI-enabled manycore architecture, an application, and a maximum allowable performance penalty, our objective is to create a DVFI control policy π^* that minimizes the energy dissipation within the user-specified maximum allowable performance penalty.

Imitation Learning Methodology

For our DVFI control problem, the expert corresponds to an Oracle controller that provides the supervision on how to make good control decisions for V/F tuning. There are three main challenges in applying the IL framework to learn DVFI control policies: 1) Oracle construction, 2) fast and accurate decision-making, and 3) learning robust control policies. We discuss and provide the corresponding solutions below.

a) Oracle Construction. In traditional IL, expert demonstrations are used as the Oracle policy in the IL process. Unfortunately, we don't have any training data for the DVFI control problem. For DVFI-enabled systems, we define the Oracle policy as the controller that selects the V/F level for each VFI that minimizes the overall power consumption within some performance constraint. Since the learning process is offline, we access the future system states and perform a look-ahead search to find the best joint V/F allocation for all VFIs. This is accomplished by running the application with different V/F assignments to optimize the global performance (i.e., EDP of the system). To overcome the computational challenge and closely approximate optimality, our *key insight is to perform local optimization followed by aggregation for global optimization.* First, we compute the optimal V/F (which minimizes EDP) for each VFI, at each control epoch, for m different execution time penalties (e.g., 0%, 5%, and 10% for m=3). This gives us m different V/F assignments for each control epoch. Second, for every n control epochs, we compute the best V/F decisions by performing an exhaustive search over all possible combinations of local optima from the first step (m^n). Note that it is easy to find a small m that works well in practice, but both the quality and computation time of the Oracle depends on n.

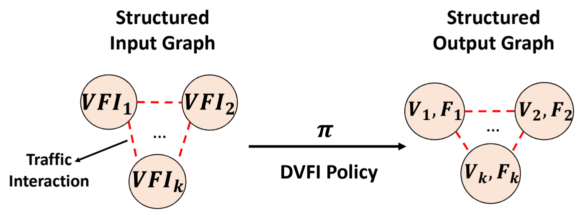


Figure 13: Illustration of DVFI decision-making as a structured prediction task

b) Fast and Accurate Decision-Making. We formulate the problem of DVFI control decision-making as a *structured output prediction* task [11, 12]. This is the task of mapping from an input structured object (a graph with features on the nodes and edges) to an output structured object (a graph with labels on the nodes and edges). Figure 13 illustrates the structured prediction task corresponding to the DVFI control decision-making. The structured input graph contains a node for each VFI and edges corresponding to inter-VFI traffic density. The structured output graph captures the V/F allocation (node labels) for each VFI and the structural dependencies between different input and output variables.

The main challenge in DVFI control is to choose the best V/F from the large space of all possible V/F assignments (L^k, where k is the number of VFIs and L is the number of V/F levels for each VFI). This is especially challenging in our DVFI control problem: we are trying to predict the joint V/F allocation for all VFIs to save energy, but it is useless if the computation for making the prediction consumes more energy than the energy saved. Therefore, we want a fast and accurate predictor whose energy overhead is miniscule when compared to the overall energy savings due to DVFI control.

To address the above-mentioned challenge, we learn *pseudo-independent structured controllers* to achieve efficiency without losing accuracy. Specifically, we learn k controllers, one controller for each VFI. These controllers are pseudo-independent in the sense that each controller predicts the V/F allocation for only a single VFI but has the context of previous predictions from all controllers and the structural dependency information of the other VFIs when making predictions. Intuitively, the different controllers are trying to help each other by supplying additional contextual information.

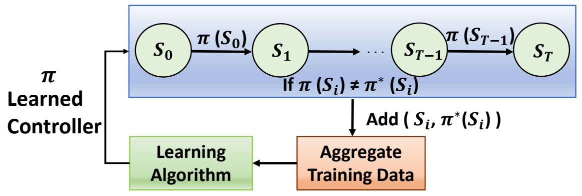


Figure 14: Illustration of Learning with DAgger

c) Learning Robust Control Policies. Our goal is to learn a controller that closely follows the Oracle in terms of V/F allocation. Unlike standard supervised learning problems that assume IID (Independent and Identically Distributed) input examples, our controller learning problem is Non-IID because the next state depends on the decision of the controller at the previous state. Therefore, controllers learned via exact imitation can be prone to error propagation: errors in the previous state may result in a next state that is very different from the distribution of states the learner has seen during the training, and contributes to more errors. To address the error-propagation problem associated with exact imitation training, we can employ an advanced imitation learning approach called DAgger [24]. The key idea behind DAgger is to generate additional training data so that the learner is able to learn how to recover from mistakes (see Figure 14).

Experimental Results

In a recent work, we showed the effectiveness of the above IL methodology when compared to the prior approaches for power management [21]. Fig. 15 shows the computational overhead for learning each DVFI policy and the full-system energy dissipation for IL, RL, and a feedback-based (FB) DVFI control policy [20]. Here, each RL and FB marker represents the results of a benchmark normalized with respect to IL. The IL policy is able to outperform every benchmark while requiring significantly less computational overhead than RL. Since there is no learning involved in FB, the

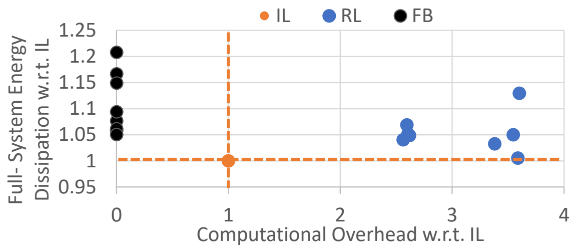


Figure 15: Comparison of several DVFI policies (IL, RL, and a feedback-based controller (FB)) in computational overhead to learn the policy and full-system energy dissipation.

computation time is negligible. The performance gap between IL methodology relative to other approaches will grow with the system size and complexity of decision-making.

5 CONCLUSIONS AND FUTURE CHALLENGES

In this work, with the focus on the on-chip network, we present an illustrative design methodology for adaptive manycore architectures to address the computational needs of emerging big data applications and the technological constraints of power and reliability. It is a cross-layer design approach that spans circuits, microarchitecture, and machine learning algorithms for intelligent and autonomous runtime adaptation.

Although the research community has made progress in exploring self-aware adaptive architectures, many important research questions remain open. Ultimately, the design of adaptive manycore architectures requires a hardware-software co-design approach for maximum benefits. In the future, adaptivity should not only expand to all components of the manycore architecture, but also operate in a holistic manner to achieve the global objectives of the system. This will involve not only highly adaptable components across the entire system (much like the ones presented in this paper), but also a detailed understanding of how each control decision or joint control decisions (e.g., DVFI, communication routing, task management, resource allocation) affects any of the system-level objectives (e.g., power, energy, latency, execution-time, and reliability constraints).

Some important research gaps include (1) dynamic allocation and reconfiguration of computing resources depending on the needs of program (e.g., the amount of parallelism in the program); (2) automatic hardware-level approximation to meet both program and system goals (e.g., execution time budget, power constraints, and resiliency) without the programming complexity of current manycore systems; (3) methodologies to enable programmers to succinctly specify the execution context along with the computational/algorithmic components of programs; and (4) innovations in control and learning techniques to handle this large complex state-action space under dynamically changing constraints with negligible computational overhead.

REFERENCES

[1] [n. d.]. Sunway TaihuLight. https://www.top500.org/system/178764. ([n. d.]).
[2] Justin A. Boyan and Andrew W. Moore. 2001. Learning Evaluation Functions to Improve Optimization by Local Search. *JMLR* 1 (Sept. 2001), 77–112. https://doi.org/10.1162/15324430152733124
[3] Amit Kumar *et al.* 2007. Express Virtual Channels: Towards the Ideal Interconnection Fabric. In *ISCA '07*. 150–161. https://doi.org/10.1145/1250662.1250681
[4] Boris Grot *et al.* 2011. Kilo-NOC: A Heterogeneous Network-on-chip Architecture for Scalability and Service Guarantees. In *ISCA '11*. 401–412. https://doi.org/10.1145/2000064.2000112
[5] Chrysostomos A. Nicopoulos *et al.* 2006. ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers. In *MICRO 39*. 333–346. https://doi.org/10.1109/MICRO.2006.50
[6] Chia-Hsin O. Chen *et al.* 2013. SMART: A single-cycle reconfigurable NoC for SoC applications. In *DATE '13*. 338–343. https://doi.org/10.7873/DATE.2013.080
[7] Daniel Johnson *et al.* 2011. Rigel: A 1,024-Core Single-Chip Accelerator Architecture. *IEEE Micro* 31, 4 (July 2011), 30–41. https://doi.org/10.1109/MM.2011.40
[8] F.A. Rezaur Rahman Chowdhury *et al.* 2017. Select-and-Evaluate: A Learning Framework for Large-Scale Knowledge Graph Search. *JMLR* 80 (2017).
[9] George Kurian *et al.* 2010. ATAC: A 1000-core Cache-coherent Processor with On-chip Optical Network. In *PACT '10*. 477–488. https://doi.org/10.1145/1854273.1854332
[10] Hiroki Matsutani *et al.* 2009. Prediction router: Yet another low latency on-chip router architecture. In *HPCA '09*. 367–378. https://doi.org/10.1109/HPCA.2009.4798274
[11] Janardhan R. Doppa *et al.* 2014. HC-search: A Learning Framework for Search-based Structured Prediction. *JAIR* 50, 1 (May 2014), 369–407.
[12] Janardhan R. Doppa *et al.* 2014. Structured Prediction via Output Space Search. *JMLR* 15, 1 (Jan. 2014), 1317–1350.
[13] Michel A. Kinsy *et al.* 2009. Application-aware Deadlock-free Oblivious Routing. In *ISCA '09*. 208–219. https://doi.org/10.1145/1555754.1555782
[14] Michel A. Kinsy *et al.* 2013. Heracles: A Tool for Fast RTL-based Design Space Exploration of Multicore Processors. In *FPGA '13*. 125–134. https://doi.org/10.1145/2435264.2435287
[15] Michel A. Kinsy *et al.* 2017. PreNoc: Neural Network Based Predictive Routing for Network-on-Chip Architectures. In *GLSVLSI '17*. 65–70. https://doi.org/10.1145/3060403.3060406
[16] Myong Hyon Cho *et al.* 2009. Oblivious Routing in On-Chip Bandwidth-Adaptive Networks. In *PACT '09*. 181–190. https://doi.org/10.1109/PACT.2009.41
[17] Natalie E. Jerger *et al.* 2017. On-Chip Networks, Second Edition. *Synthesis Lectures on Computer Architecture* (2017).
[18] Pengju Ren *et al.* 2016. A Deadlock-Free and Connectivity-Guaranteed Methodology for Achieving Fault-Tolerance in On-Chip Networks. *IEEE TC* 65, 2 (Feb. 2016), 353–366. https://doi.org/10.1109/TC.2015.2425887
[19] Pengju Ren *et al.* 2016. Fault-Aware Load-Balancing Routing for 2D-Mesh and Torus On-Chip Network Topologies. *IEEE TC* 65, 3 (March 2016), 873–887. https://doi.org/10.1109/TC.2015.2439276
[20] Ryan G. Kim *et al.* 2016. Wireless NoC and Dynamic VFI Codesign: Energy Efficiency Without Performance Penalty. *IEEE TVLSI* 24, 7 (2016), 2488–2501.
[21] Ryan G. Kim *et al.* 2017. Imitation Learning for Dynamic VFI Control in Large-Scale Manycore Systems. *IEEE TVLSI* 25, 9 (Sept 2017), 2458–2471. https://doi.org/10.1109/TVLSI.2017.2700726
[22] Sourav Das *et al.* 2017. Design-Space Exploration and Optimization of an Energy-Efficient and Reliable 3-D Small-World Network-on-Chip. *IEEE TCAD* 36, 5 (May 2017), 719–732. https://doi.org/10.1109/TCAD.2016.2604288
[23] Sunghyun Park *et al.* 2012. Approaching the theoretical limits of a mesh NoC with a 16-node chip prototype in 45nm SOI. In *DAC '12*. 398–405.
[24] Stéphane Ross *et al.* 2011. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *AISTATS '11*, Vol. 15. Fort Lauderdale, FL, USA, 627–635.
[25] Tushar Krishna *et al.* 2013. Breaking the on-chip latency barrier using SMART. In *HPCA '13*. 378–389. https://doi.org/10.1109/HPCA.2013.6522334
[26] Umit Y. Ogras *et al.* 2009. Design and Management of Voltage-frequency Island Partitioned Networks-on-chip. *IEEE TVLSI* 17, 3 (2009), 330–341.
[27] Hwisung Jung and Massoud Pedram. 2010. Supervised Learning Based Power Management for Multicore Processors. *IEEE TCAD* 29, 9 (Sept. 2010), 1395–1408.
[28] Monodeep Kar and Tushar Krishna. 2017. A Case for Low Frequency Single Cycle Multi Hop NoCs for Energy Efficiency and High Performance. In *ICCAD '17*. IEEE.
[29] Michel A. Kinsy and Srinivas Devadas. 2014. Low-overhead hard real-time aware interconnect network router. In *HPEC '14*. 1–6. https://doi.org/10.1109/HPEC.2014.7040976
[30] Jan M. Rabaey and Anantha Chandrakasan. 2002. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall Pub.