Memory-Access Aware DVFS for Network-on-Chip in CMPs

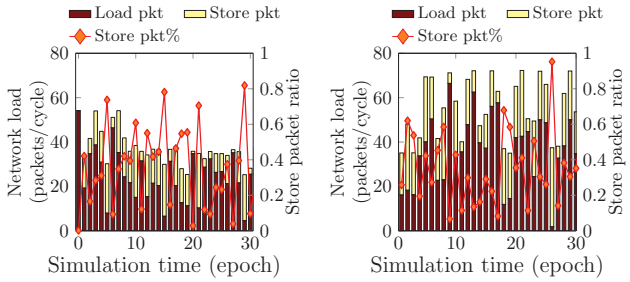
Yuan Yao and Zhonghai Lu
KTH Royal Institute of Technology, Stockholm, Sweden
{yuanyao, zhonghai}@kth.se

***Abstract*—We present a new DVFS technique for network-on-chip (NoC) that adjusts the voltage/frequency scales of routers according to memory-access characteristics of application running on the CMP. The memory characteristics are periodically profiled, reflecting both resource-access density in the network and memory-access criticality for application performance. The network conducts per-router voltage/frequency tuning using the memory-access density information while it performs priority-based switch allocation to speed up critical packets and avoid starvation using the memory-criticality information. Compared to a latest per-router DVFS approach, benchmark experiments demonstrate that our memory-access characteristics aware DVFS technique achieves not only better power saving, energy-delay product, but also enhanced network and application performance.**

I. Introduction

Due to the dependency of dynamic power on voltage and frequency, the power efficiency of NoC can be improved by Dynamic Voltage and Frequency Scaling (DVFS) techniques [4], [8]. To identify the optimal V/F setting, the majority of recent works in NoC DVFS rely on metrics dynamically gathered from the network. Although such works demonstrate the benefits of workload-sensitive DVFS in the NoC, the exploited metrics only reflect network performance, which are not directly related to application-level metrics such as memory-access density and memory-access criticality.

To exemplify our point, we investigate dynamic memory access characteristics of multi-threaded workloads in PARSEC [2] under the Gem5 [3] simulation framework (see detailed system configuration in Table 1 of Section 5). Fig. 1 presents two applications, *blackscholes* (memory non-intensive application) and *canneal* (memory-intensive application), with each bar denoting the simulation results averaged over 1 million simulation cycles (called an epoch). Based on the figure, we can make the following observations. **1)** Because *blackscholes* is memory non-intensive, it generates less network loads than *canneal* and has greater potential for making fast progress in the processor. Because such memory non-intensive application is often more sensitive to memory access stall time and network delay [7], decreasing the network V/F level under *blackscholes* may notably decrease its application-level performance. **2)** Under memory-intensive program *canneal*, it may not always be appropriate to increase the V/F level under high network load. This is because such programs may incur non-proportional load/store memory access ratio (as shown in Fig. 1(b)). Although many store packets induce large network load, they might not be critical to the overall application performance, since their delay can be tolerated by various



(a) *Blackscholes* (b) *Canneal*

Fig. 1: Dynamic memory access characteristics of representative PARSEC benchmarks

delay hiding techniques such as read/write buffers [9]. Hence, having no knowledge of both memory-access density and criticality may result in inefficient NoC power provisioning and unsatisfactory NoC DVFS.

The paper proposes a memory-access characteristics aware DVFS technique for NoCs in CMPs. We develop a three-parameter model, $(\rho, \gamma_{L1m}, \gamma_{load})$, to reflect a core's runtime memory-access characteristics and guide the per-router V/F adjustment, where ρ represents a core's network access rate measured in averaged number of entries in per-core miss status holding registers (MSHRs), γ_{L1m} for the private L1 cache miss per instruction (MPI), and γ_{load} for the load instruction ratio. While ρ captures the memory-access density in the network, γ_{L1m} and γ_{load} reflect the memory-access criticality. The network interface (NI) characterizes these parameters and send them to the network periodically. The router uses ρ from cores to tune the V/F setting and the other two parameters to improve application performance and avoid starvation. The experiments show that our approach can improve power efficiency and system-level EDP (energy-delay product) over a mechanism emulating a state-of-the-art DVFS technique in [12].

II. Related work

Many network-level metrics have been exploited to implement DVFS in NoCs. In [6], the entire NoC is considered as a single V/F domain to offer coarse-grain DVFS management. A centralized PI (proportional and integral) controller is used to dynamically adjust the V/F scale according to network throughput and average packet delay. To capture the impact of the uncore upon overall system performance, [5] proposes a DVFS approach using average network delay and memory access time as the V/F adjustment hints. Work in [4] illustrates that computational workloads in multicores are highly complex and exhibit fractal characteristics. A characterization approach

978-3-9815370-6-2/DATE16/

based on the dynamism of queue utilization and fractional differential equations has been proposed to guide on-chip DVFS in different V/F islands. In [12], Mishra *et al.* propose a per-router DVFS approach for NoC. They monitor router input queue occupancy, based on which the router changes its V/F level. Further in [8], per-router DVFS has been shown to achieve efficient V/F adjustment in which the overall V/F scaling delay can be limited to tens of processor cycles.

III. Memory-Access Profiling

A. Design Overview

In CMP, the NoC typically interconnects processor nodes, which contain a CPU core with its private L1 cache, shared on-chip last-level cache banks, and possibly on-chip memory controllers, as shown in Fig. 2. We assume that applications running on the CMP are multithreaded. Different threads execute concurrently, with one core running one thread. Each core is connected to the network via its network interface (NI), and cores communicate memory-access and cache coherence messages with each other via the network. Our memory-access profiling mechanism is realized in the NI, where a characterizer implements the memory-access characterization functions. The NI is further responsible for packetization and sending the memory-access characteristics to the network via piggybacking packets.

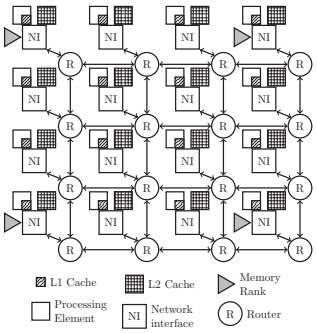


Fig. 2: A 4×4 mesh CMP

B. Memory-Access Characterization

To accurately capture per-core runtime memory-access characteristics, we introduce a three-parameter $(\rho,\ \gamma_{L1m},\ \gamma_{load})$ model, where ρ represents a core's memory-access density measured in average network requests in MSHRs, γ_{L1m} and γ_{load} jointly reflect a core's memory-access criticality, where γ_{L1m} the private cache misses per instruction, and γ_{load} the load instruction ratio among all load/store instructions. Note that, since the parameters are defined to profile the actual traffic to the NoC, only those cache misses and load instructions that incur network messages are counted.

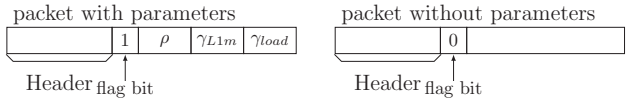
To allow continuously characterize these parameters with a proper time granularity, we further use a sliding-window based characterization [11]. A sliding-window is a time interval of length T_{sw} by which the parameters are characterized or sampled. After each characterization, the window slides forward one epoch T (in the paper, we set T equals to $T_{sw}/4$).

Characterization of $\boldsymbol{\rho}$: In our model, ρ represents the per thread memory-access density measured in average number of network requests per time window.

Characterization of $\boldsymbol{\gamma_{L1m}}$: In our model, private cache miss per instruction (MPI), γ_{L1m}, intends to capture the memory access criticality of a thread. The insight is straightforward: an application with a small number of L1 misses is likely to issue relatively less network requests. Hence prioritizing this application's requests allows the application's running core to make fast progress without experiencing heavy network delay.

Characterization of $\boldsymbol{\gamma_{load}}$: State-of-the-art CMPs utilize load/store queues (LSQ) to implement memory disambiguation [9], in which the load instructions are more critical than store instructions, since they may cause a core to stall due to NoC transmission delay. In our model, we use load instruction ratio to reflect this kind of memory-access criticality.

C. Piggybacking Parameter Propagation



(a) Packet with parameters (b) Packet without parameters

Fig. 3: Piggybacking the parameters in packet

The three per-core characterized parameters $(\rho,\ \gamma_{L1m},\ \gamma_{load})$ have to be spread into the network in order to guide the router V/F-level adjustment. We propose a distributed *piggybacking* parameter-propagation approach in two steps: 1) We treat the parameters as payload, part of the first message in each packet route. During each characterization window, the first packet of the first message in each packet route carries the parameter set during packetization. As shown in Fig. 3, specific bits in a packet format are added to indicate if the packet carries the parameters. 2) When the first packet is routed in the network, the passing routers record the parameters upon detecting the packet header. As messages are continuously sent to the network, the parameters are updated window by window. Note that the parameter-payload detection can be conveniently done. Since an on-chip router distinguishes head, body, tail flits for each packet at the flow control level, we only need to check the flag bit in the head flit to determine if the following constant-size bits of the packet carry the parameters.

IV. Router Architecture

A. Design Overview

We present a pipelined router supporting memory-access aware DVFS. Fig. 4 sketches the micro-architecture of the proposed router partitioned into data path and control path. Our baseline design is a 2-stage speculative router [13] in which the first stage performs Route Computation (RC), Virtual Channel (VC) Allocation (VA), and Switch Allocation (SA) in parallel and the second stage is Switch Traversal (ST).

The highlighted components in Fig. 4 cooperatively achieve our goals. The Communication Demand Computation logic (CDC) computes the communication demand according to core memory-access density ρ. The DVFS setting logic ensures that the router operates at the lowest V/F level but still meets the memory-access density requirement ρ. To alleviate the performance penalty suffered by memory non-intensive and high load-instruction ratio applications under low V/F levels, our router implements priority-based switching, which periodically forms an ordering of departure packets. The packets departing order is determined by using r_{L1m} and r_{load}

as heuristics which capture the memory-access criticality of each application.

B. Design Details

Communication Demand Computation (CDC). Assuming router i has n ports, each of which consists of m virtual channels. The total communication demand from cores at router i can be calculated by:

$$demand_i = \sum_{j=1}^{n} max(\rho^{j,k}), \quad (1)$$

where $k = 1, 2, ...m$. $demand_i$ reflects how many memory requests that router i must transmit per cycle in order to avoid congestion. The maximum value of $demand_i$ is m (given the worst case that each router input port receives one memory request per cycle) and the minimum 0. The CDC is added to the first pipeline stage of the router, running in parallel with RC, VA and SA. Since it contains only simple addition and comparison operations, it does not increase the critical path.

DVFS setting logic. A periodically reset counter counts the clock cycles to achieve the timeframe-based DVFS setting. Considering the V/F adjustment overhead, it is not always necessary to update the V/F setting once the $demand_i$ is changed across different DVFS tuning frames. Rather than that, we divide the frequency span (maximum frequency - minimum frequency) of router i evenly into l levels, with each $demand_i$ mapped into one level. During program runtime, the V/F setting logic checks the $demand_i$ at the start of each frame. Only when $demand_i$ shifts into another level, the V/F tuning procedure is performed. Otherwise, the upcoming V/F tuning frame inherits the same V/F level from the previous frame. In our current implementation, we support 3 different V/F configurations (1.5V/1.5GHz, 1.6V/1.75GHz, 1.7V/2.0GHz), and we evenly divide $demand_i$ into 3 segments with each segment mapped to one V/F configuration in the set.

Priority-based switch allocator. To achieve high application performance, our DVFS mechanism further adopts a *packet ranking* mechanism. Each router prioritizes packets according to the following rank: packets of a memory non-intensive application, or packets of a high load-instruction ratio application are prioritized over others. Towards this end, the switch allocator periodically forms an order of departure packets, which is determined by using γ_{L1m} and γ_{load} as the heuristics to capture the memory access criticality of an application. To prevent starvation, we combine the prioritization based switch allocation with a *packet batching* mechanism as introduced in [7]. Each packet is added to a batch depending on its time of arrival, and packets belonging to previous batches are prioritized over packets from later batches. For implementation simplicity, we set the memory-access characterization window length and the packet batch length both equal to the length of a DVFS timeframe.

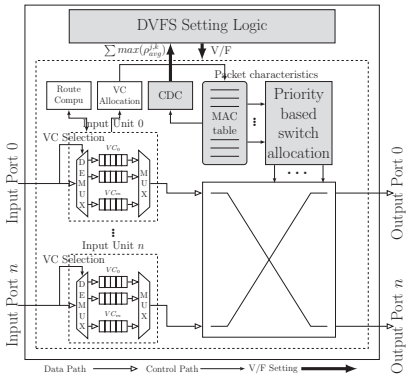


Fig. 4: The V/F regulated router architecture

Memory-Access Characteristics (MAC) table. To support the CDC and priority-based switch allocation, a router must record the three parameters, $(\rho, r_{L1m}, r_{load})$, carried by the first message of different routes during each characterization window. This is achieved by using a MAC table which is organized as follows: there is one table associated with each router, and one entry corresponds to one VC. If multiple VCs are occupied by the same application/thread, they share the same $(\rho, r_{L1m}, r_{load})$ values and are treated equally in the switch allocation. Since these values are not carried by all packets, the MAC table is updated epoch by epoch.

V. Experiments and Results

A. Methodology

Experimental setup. We integrate our design with the cycle-accurate simulator Gem5 [3], in which the built-in network Garnet [1] implements the new NI and router architecture supporting our DVFS mechanism. The simulation platform configurations are shown in Table I. To evaluate our design with different schemes, we experiment with the region-of-interest of real benchmark programs PARSEC [2]. To scale PARSEC well to the target 64-core CMP, we select large input sets (`simlarge`) for all benchmarks. Further, we utilize Orion 2.0 [10] to report the power related results.

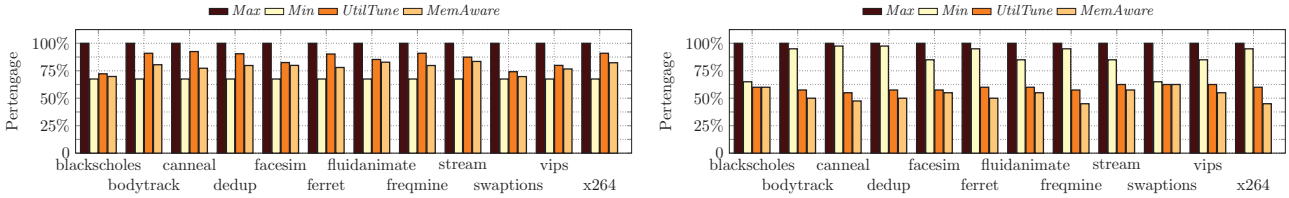
Comparison studies. For comparison purpose, we implement a memory-access oblivious DVFS mechanism emulating closely the technique in [12], in which the NoC allows per-router V/F adaption by monitoring the router *buffer utilization* and adjusts V/F scales according to empirical thresholds. When the per frame average buffer utilization exceeds 0.6 packets/cycle, the router can adaptively switch to the high V/F mode (1.7V/2.0GHz) to accelerate packet transmission. Otherwise, when the average buffer utilization is below 0.4 packets/cycle, the router switches to low V/F mode (1.5V/1.5GHz) for power saving. We name this DVFS mechanism *UtilTune* in our experiments.

B. Benchmark Results

Application-level power consumption and energy delay product (EDP). Fig. 5(a) and 5(b) show the power consumption and energy delay product (EDP). *MemAware* represents our memory-access characteristics aware DVFS. *Min* depicts a network which runs at minimum V/F level (which is 1.5V/1.5GHz and is also the start V/F in *MemAware* and *UtilTune*) and *Max* at maximum V/F level (1.7V/2.0GHz). In each benchmark, the results are normalized with respect to *Max*, which consumes the most power due to constantly high router frequency. **In terms of power consumption**, *Min* spends less power than *UtilTune* and *MemAware* across all the PARSEC benchmarks, which is mainly caused by

TABLE I: Simulation platform configuration

Item	Amount	Configuration
Processor	64 cores	Alpha based 2.0 GHz out-of-order processors. 32-entry instruction queue, 64-entry load/store queue, 128-entry reorder buffer (ROB).
L1-Cache	64 banks	Private, 32KB per-core, 4-way set associative, 128B block size, 2-cycles latency, split I/D caches, 32 MSHRs.
L2-Cache	64 banks	Chip-wide shared, 1MB per-bank, 16-way set associative, 128B block size, 6-cycles latency, 32 MSHRs.
Memory	8 ranks	4GB DRAM, 512MB per-rank, up to 16 outstanding requests for each processor, 8 memory controllers.
NoC	64 nodes	8×8 mesh network. Each node consists of 1 router, 1 network interface (NI), 1 core, 1 private L1 cache, and 1 shared L2 cache. X-Y dimensional routing. Each router has 6 VCs per port, 4 flits per VC. 128-bit datapath. Directory based MOESI cache coherence protocol. One cache block consists of 1 packet, which consists of 8 flits. One coherence control message consists of 1 single-flit packet.
DVFS ctrl.	1/router	16,384 (2^{14}) cycles of DVFS tuning timeframe (the same size of a thread memory-access characterization window and a packet prioritization batch). 3 supported V/F levels: 1.5V/1.5GHz, 1.6V/1.75GHz, 1.7V/2.0GHz.



(a) Power consumption results

(b) EDP results

Fig. 5: Power consumption and EDP results across PARSEC benchmarks

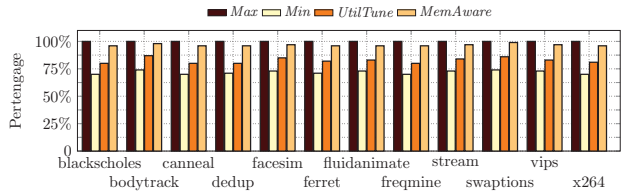


Fig. 6: System IPC results across PARSEC benchmarks

the constantly lower router voltage/frequency. As shown in Fig. 5(a), the power saving of *MemAware* against *UtilTune* depends on the characteristics of the benchmarks. With high network utilization benchmarks (*bodytrack*, *canneal*, *dedup*, *ferret*, *freqmine*, and *x264*), the efficiency of *MemAware* against *UtilTune* becomes more obvious. At higher network traffic, when more threads competing for network resources, *MemAware* assigns per-router V/F based on the router's demand, which achieves more efficient power consumption. The best case happens at *canneal*, where *MemAware* consumes 17% less power than *UtilTune*. With low network utilization benchmarks (*blackscholes* and *swaptions*), *Min*, *UtilTune* and *MemAware* deliver similar performance. This is because that most of the time the NoC remains under-loaded, and the DVFS policies have little effects on power saving. **In terms of EDP,** which considers both performance and energy, *Min* and *Max* do not perform better than *UtilTune* and *MemAware*. Further, as shown in Fig. 5(b), *MemAware* gives better EDP in all benchmarks (with best case happens in *freqmine*, where *MemAware* achieves 19% EDP reduction than *UtilTune*) but two low network utilization ones, where EDP of *MemAware* is close to that of *UtilTune*. In summary, although *MemAware* consumes slightly more power than *Min*, it delivers better overall EDP than the other three schemes.

Instructions Per Cycle (IPC) results. Fig. 6 presents the system-level IPC results. The figure shows *MemAware* consistently achieves better IPC than *UtilTune* and *Min*, but the gain depends on the network load and traffic pattern created by each application. As we can observe, applications with a larger network load receives higher benefits from *MemAware*. With *canneal*, *MemAware* achieves largest improvement in system IPC, 16% against *UtilTune* and 26% against *Min*.

VI. Conclusion

The paper presents a memory-access characteristics aware DVFS technique for on-chip networks. We compare our approach with a memory-access oblivious per-router DVFS scheme named *UtilTune* in the paper. With PARSEC benchmark programs, our proposed DVFS achieves up to 17% reduction in power consumption against *UtilTune*. In terms of EDP, which considers both performance and energy, our DVFS gives better performance in all the benchmarks, with the best case achieving 19% EDP reduction than *UtilTune*. As for IPC, our technique outperforms *UtilTune* in all the benchmarks with largest improvement of 16%. In the future, we plan to study the thermal effect of our DVFS scheme.

References

[1] N. Agarwal, T. Krishna *et al.*, "GARNET: A Detailed On-chip Network Model Inside a Full-system Simulator," in *ISPASS*, 2009.

[2] C. Bienia, S. Kumar *et al.*, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Aug. 2008.

[3] N. Binkert, B. Beckmann *et al.*, "The Gem5 Simulator," *SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[4] P. Bogdan, R. Marculescu, and S. Jain, "Dynamic Power Management for Multidomain System-on-Chip Platforms: An Optimal Control Approach," in *TODAES*, 2013.

[5] X. Chen, Z. Xu *et al.*, "In-network Monitoring and Control Policy for DVFS of CMP Networks-on-Chip and Last Level Caches," *International Symposium on Network on Chip (NoCS)*, 2012.

[6] ——, "Dynamic Voltage and Frequency Scaling for Shared Resources in Multicore Processor Designs," in *DAC*, Jun. 2013.

[7] R. Das, O. Mutlu *et al.*, "Application-aware Prioritization Mechanisms for On-chip Networks," in *International Symposium on Microarchitecture (MICRO)*, Jul. 2009.

[8] S. Eyerman and L. Eeckhout, "Fine-grained DVFS Using On-chip Regulators," *ACM Transations on Architecture and Code Optimization (TACO)*, 2011.

[9] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach, Fifth Edition*. MK. Publishers Inc., 2011.

[10] A. B. Kahng, B. Li *et al.*, "ORION 2.0: A Power-Area Simulator for Interconnection Networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 191–196, Jan. 2009.

[11] Z. Lu and Y. Wang, "Dynamic Flow Regulation for IP Integration on Network-on-Chip," in *NoCS*, 2012.

[12] A. K. Mishra, R. Das *et al.*, "A Case for Dynamic Frequency Tuning in On-chip Networks," in *MICRO*, Jul. 2009.

[13] L.-S. Peh and W. J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers," in *International Symposium on High-Performance Computer Architecture (HPCA)*, 2001.