Energy-Efficient Non-Minimal Path On-chip Interconnection Network for Heterogeneous Systems

Jieming Yin*, Pingqiang Zhou†, Anup Holey*, Sachin S. Sapatnekar†, and Antonia Zhai*

*Department of Computer Science and Engineering
University of Minnesota, Twin Cities
Minneapolis, Minnesota 55455, USA
{jyin, aholey, zhai}@cs.umn.edu

†Department of Electrical and Computer Engineering
University of Minnesota, Twin Cities
Minneapolis, Minnesota 55455, USA
{pingqiang, sachin}@umn.edu

ABSTRACT

Network-on-Chips (NoCs) in heterogeneous systems containing both CPU and GPU cores must be designed to satisfy the performance requirements of both latency-sensitive CPU traffic and throughput-intensive GPU traffic. DVFS and adaptive routing can potentially improve the NoC efficiency. We further notice that GPU traffic can sometimes tolerate a slack defined as the number of cycles a packet can be delayed without causing performance penalty. In this work, we take advantage of the slack in GPU packets to route packets through non-minimal path, so that routers can operate at a lower frequency without suffering performance penalty.

Keywords

Heterogeneous Multi-core, Interconnects, NoC, Voltage Scaling, Frequency Scaling, Non-minimal Path Routing.

1. INTRODUCTION

CPU cores nowadays are optimized for single-thread workloads, while GPU cores are optimized for high throughput workloads. Heterogeneous multi-core systems that contain both CPU and GPU cores can potentially achieve the best of both worlds in an energy efficient manner by utilizing different cores as workload characteristics change [20]. In a heterogeneous multi-core system, different components on the die must communicate through a network on-chip (NoC). Since the NoC is the critical resource shared by all the applications running on the processor, it must be carefully designed to satisfy the energy envelop and to meet the requirements for variety of traffic patterns.

In a modern CPU, the processor stalls when requested data are not returned. Therefore, traffic originated from or designated to a CPU core is latency-sensitive. Applications that are computation-bound generate relatively little NoC traffic; while data centric applications can generate significant traffic. The former has a moderate throughput requirement; and the latter has a considerable throughput requirement.

A GPU consists of a collection of data-parallel compute cores referred to as streaming multiprocessors (SMs). A large number of lightweight threads are allocated to each SM and are batch-scheduled on the SIMD pipeline in a fixed size group called a warp. In other words, warps are the basic execution units in a GPU. All the threads within a warp execute the same instruction but operate with different data values. When a thread in a particular warp encounters a load miss, the entire warp is context-switched out, and another available warp is scheduled. Notice that usually there are hundreds of threads executing on a GPU in parallel, so it is highly possible that large numbers of outstanding memory requests are being processed in the interconnection network. Thus GPU traffic requires high network bandwidth. On the other hand, as long as there are enough available warps to schedule in an SM core, the system performance would not be harmed despite the long memory access latency, since the latencies are hidden by scheduling other ready warps. As a result, some GPU traffic is latency-insensitive, and can be delayed in the NoC without performance impact.

Dynamic Voltage and Frequency Scaling (DVFS) is an effective technique for reducing network energy dissipation [12, 17, 21]. The key idea behind network DVFS is to dynamically scale the supply voltage of the routers and links to provide *just-enough* circuit speed to process the traffic workload. Reducing NoC frequency can increase latency and/or reduce throughput, thus can only be applied to workloads with moderate latency and throughput requirements. However, in heterogeneous systems where traffic pattern is mixed, it is difficult to determine the optimal DVFS level.

In this work, we notice that GPU traffic can sometimes tolerate a slack defined as the number of cycles a packet can be delayed without causing performance penalty. Taking advantage of the slack in GPU packets, we propose an NoC design to route packets through non-minimal paths. Such design balances on-chip traffic workloads so that routers can operate at a lower frequency without suffering significant performance penalty.

This paper makes the following contributions:

- We observe that the number of available warps in an SM directly relates to the delay introduced in warp scheduling. Taking advantage of this observation, we propose a mechanism that can dynamically predict the network packet slack. This slack predicting mechanism is applied to GPU messages.
- We propose an energy-efficient network tuning policy that consists of a DVFS mechanism as well as a non-minimal path routing algorithm to reduce the energy consumption of the on-chip interconnection network while minimizing the performance degradation caused by network frequency scaling.

2. EXPLOITING SLACK IN GPU

Based on traffic characteristics, different strategies should be applied to different network messages. For CPU applications, a reduction in NoC throughput has relatively little impact on overall performance, but increasing NoC latency can cause significant performance degradation. While for GPU applications, delaying memory requests in the network has little effect on performance, provided a minimum network throughput is guaranteed. However, we cannot increase the NoC latency of a GPU packet limitlessly. Hence, it is necessary to find an upper bound for delaying a network packet for GPU applications.

Since all outstanding memory requests are sent via the on-chip interconnection network in the form of packets, we first define the *slack of a packet* as the number of cycles a packet can be delayed without affecting the overall execution time.

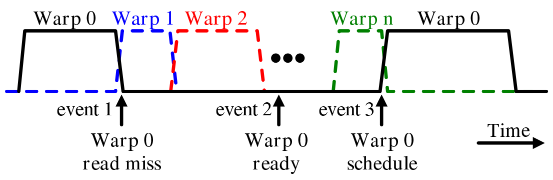


Figure 1: Execution of a GPU warp and packet slack

Figure 1 demonstrates the execution of a GPU warp (Warp 0) and its packet slack. In an SM, a warp is available if all operands are ready. Assume we have n available warps in an SM that can be scheduled at a given time; Warp 0 is scheduled and executed first. After a few cycles of execution, a read miss occurs, which is denoted as Event 1 in the figure. Then a hardware context switch is performed and Warp 1 gets scheduled and executed. Similarly, when Warp 1 encounters a read miss, the next ready warp is scheduled accordingly. Event 2 represents the cycle when the memory request of Warp 0 is sent back, which means Warp 0 is ready to schedule. However, depending on the scheduling algorithm, Warp 0 may not be scheduled instantly since there are other ready warps waiting in the scheduling queue in front of Warp 0. After a read miss of the last available warp, Warp 0 is rescheduled at Event 3. Packet slack for Warp 0 is the time between Event 2 and 3. It is evident that if we delay the network transmission of the packet for a number of cycles smaller than the slack, the overall execution time will remain unaffected.

Intuitively, packet slack is related to the number of available warps ready to schedule in an SM. The more available warps an SM has during a period of time, the longer the slack it can exploit for each packet. However, the number of available warps within each SM varies from application to application. To further understand the relation between packet slack and the number of available warps, we manually delay the scheduling of each warp for a fixed number of cycles after it is ready. For example, in a 10-cycle-delay experiment, if a warp is ready to be scheduled at cycle t_0, we arbitrarily prevent it from scheduling until cycle t_1=t_0+10 even if there is no warp available to schedule in between. In the worst case, execution is stalled and system performance is degraded due to the arbitrary delay.

Figure 2 shows the performance degradation due to the arbitrary delay of warp scheduling. We notice that for some of the GPU applications such as BLACKSCHOLES, large schedule delay can be tolerated. By contrast, for applications such as LIB, even a few cycles of delay will result in significant performance degradation. Our hypothesis is that delay tolerable applications have more warps allocated in each SM. To verify our hypothesis, we collect the average number of available warps for each application during the whole execution as shown in Figure 3. Here we set a 95% system speed up as the acceptable performance degradation level. The *tolerable delay cycles* is the maximum number of cycles a particular warp scheduling can be delayed without exceeding the 95% speed up threshold. From the figure, we observe that there is a linear relationship between the average number of available warps and tolerable delay cycles for all the GPU applications.

Based on the observation, we predict the approximate slack of a packet as the number of available warps in the SM in a given cycle. An additional register (*Avail_warp_reg*) which keeps track of the number of available warps for each SM is added in our design. When a packet is sent from/to a particular SM core, the *Avail_warp_reg* is referred and the packet slack is set according to the value register. When possible, packets with slack can be sent through non-minimal paths to less congested routers, balancing the traffic and alleviating congestion in busy routers.

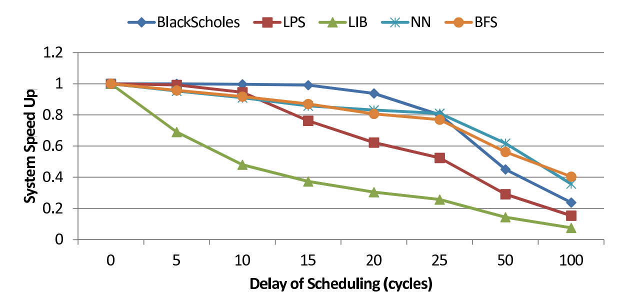


Figure 2: System performance degradation due to delay of warp scheduling

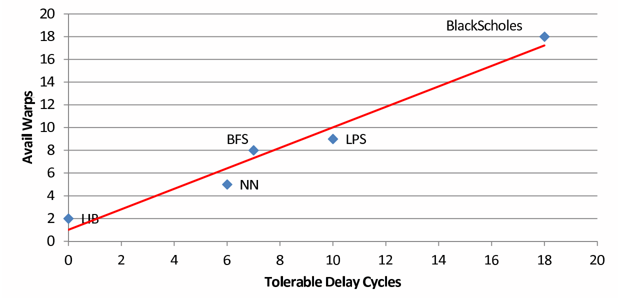


Figure 3: Linear relationship between number of available warps and tolerable delay cycles

3. DYNAMIC NETWORK TUNING POLICY

We enhance *DVFS* and propose *Non-Minimal Path Adaptive Routing* that work interactively to balance network traffic. In particular, the former periodically tunes the router supply voltage as well as the operation frequency based on the NoC workload. The latter takes into consideration the router utilization in conjunction with packet slack allowance, and *selectively* routes packets through non-minimal paths, which can facilitate more aggressive DVFS. By working in tandem, these two mechanisms are able to balance the traffic across the entire NoC and in turn, minimize global network energy consumption.

3.1 Baseline Router Architecture

To avoid the latency increase of CPU traffic when passing through routers operating at lower frequency, we adopt the flexible-pipeline router design proposed in [23]. Voltage and frequency scaling (VFS) reduces router energy consumption at the cost of increase in packet transmission time, and degradation in throughput. In flexible-pipeline routers, to maintain a constant latency through routers upon VFS, pipeline stages are reconfigured. However, flexible-pipeline routers cannot avoid the throughput degradation associated with VFS. Thus, traffic designated to cores with high throughput requirement are dynamically rerouted to avoid contention. This is achieved through *DVFS* and *Non-Minimal Path Adaptive Routing* as described in the remainder of this section.

Based on a classic four-stage-pipelined virtual-channel (VC) router with p input/output ports, as used in Garnet [2], providing different supply voltages, flexible-pipeline routers can be operated at three different modes: four-stage-, two-stage-, and single-stage-pipeline modes. We follow the notation of *slow down parameter S* introduced in [23]. A slow down parameter indicates the router operating frequency. For example, S=1 means the router is working at the default frequency without slowing down; S=2 implies that the particular router is operating at half of the default frequency, while S=4 implies that the router is operating at a quarter of the full speed. Correspondingly, a full-speed router works in four-stage-pipeline mode; half-speed router works in two-stage mode; and quarter-speed router works in single-stage mode.

3.2 DVFS Mechanism

The router operation frequency is scaled periodically based on its utilization in comparison to two thresholds, $Threshold_{High}$ and $Threshold_{Low}$. Over a reconfiguration period T, router utilization (μ) is defined as the percentage of active cycles. At the end of each period, if μ exceeds $Threshold_{High}$, the supply voltage and router frequency are scaled up; if μ is below $Threshold_{Low}$, they are scaled down. After a few reconfiguration periods, the network becomes stable without drastic phase changes. And on-chip traffic is as well balanced. While other metrics can also be used to determine the router operation level [17], router utilization is demonstrated effective for our purposes. A thorough evaluation of DVFS metrics is beyond the scope of this work.

Scaling voltage and frequency incurs performance overhead: all existing packets in a router must be cleaned up before VFS; and it takes time to reconfigure circuitry upon VFS, modeled as a 100-cycle delay in our evaluation. The reconfiguration period must be sufficiently long to amortize this overhead. On the other hand, an unreasonably long T limits the opportunities for capturing and responding to real-time traffic patterns. In our experiment, T is set to 20,000 cycles to ensure that the overhead is under 1%.

Between $Threshold_{High}$ and $Threshold_{Low}$ is the stable state where supply voltage and router frequency remain unchanged. If the range is too wide, NoC is unable to fully exploit the potential of DVFS; if it is too narrow, router voltage/frequency toggle frequently, resulting in increasing overhead. In our evaluation, these two thresholds are empirically determined: $Threshold_{High}$=0.6 and $Threshold_{Low}$=0.4.

3.3 Non-Minimal Path Adaptive Routing

This section describes an adaptive routing algorithm allowing packets that can tolerate slack to route through non-minimal paths to reduce NoC energy consumption. Although routing packets through longer paths leads to a longer per-packet delay, targeting only latency-tolerable packets can minimize the performance impact. Meanwhile, non-minimal path routing exploits under-utilized routers and avoids congested ones to balance workload across the NoC, which in turn enables further DVFS. As a result, the NoC is able to meet the desired performance level with less overall energy consumption.

A divergence from the minimal path is referred to as a *Detour*. A detour adds two extra hops to the length of the minimal path. In the absence of contention, each detour adds 10 cycles to the latency of a packet: 8 cycles for traversing through two four-stage-pipeline routers, and 2 cycles for traversing through the two additional links. Thus, packets must be able to tolerate at least a 10-cycle delay to be considered for one detour. Slack level (SL) describes the number of detours a packet is eligible for. For example, packets that can tolerate a longer than 20-cycle delay has a SL of two. Each packet injected into the network is associated with a slack level. The SL estimation of GPU packets is described in Section 2, whereas CPU packets SL is set to zero because CPU messages are latency-sensitive. In the routers, the SL of a packet is decremented once it is detoured.

Figure 4 illustrates the benefit of non-minimal path routing. In this example, routers in the top row are under-utilized and are operating at a low frequency; while those in the bottom row operate at a high frequency. This is because a large number

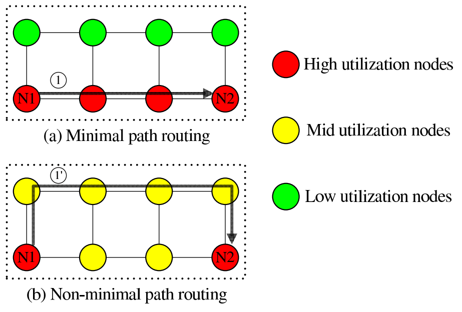


Figure 4: Non-minimal path routing

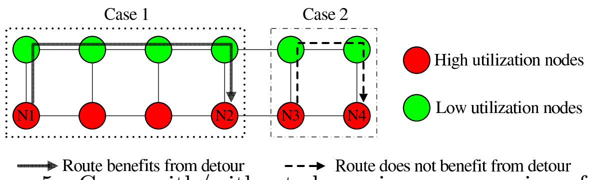


Figure 5: Cases with/without dynamic energy savings from non-minimal path routing

of packets are sent from $N1$ to $N2$, while traffic among other nodes is insignificant. In Figure 4(a), packets are sent from $N1$ to $N2$ via the minimal path. They traverse through four highly utilized routers. In Figure 4(b), some packets are sent via the non-minimal path and traverse through a longer route with two highly utilized routers and four under-utilized routers.

Let's consider the impact of DVFS, assuming the NoC can operate at three separate voltage/frequency levels: `high` (voltage = 1.2V; frequency = 1.5GHz), `medium` (voltage = 1.0V; frequency = 0.75GHz) and `low` (voltage = 0.8V; frequency = 0.375GHz). In Figure 4(a), four routers operate at high voltage/frequency level and four operate at the low level; while in Figure 4(b), two routers operate at high voltage/frequency level, and the remaining six operate at medium level. Network energy is categorized into dynamic, static, and clock energy, where static and clock energy correspond to a significant potion of the total energy [10, 23]. Static energy is proportional to supply voltage (V_{dd}), and clock energy is proportional to the product of V_{dd}^2 and number of clock transitions. Compared to routers operating at the high voltage/frequency level, clock and static energy consumption is reduced by 45% when operating at medium level, and by 65% when operating at low level. As a result, in the aforementioned example, allowing non-minimal path routing enables saving in static and clock energy.

Packets routed through non-minimal path can increase dynamic energy consumptions by traversing through additional hops, however they can also reduce dynamic energy consumptions by avoiding routers operating at high frequency. Thus, an important factor that determines whether a detour saves dynamic energy is the distance between the node where detour takes place and the packet's destination. As shown in Figure 5 Case 2, consider a packet sent from $N3$ to $N4$. Routing these packets through a non-minimal path does not provide any benefit in terms of dynamic energy savings. Whereas in case 1, dynamic energy savings is possible since packets traverse through four low frequency nodes and two high frequency nodes through a non-minimal path rather than four high frequency nodes. In our experiment, the minimum displacement towards either x or y dimension between the detour node and packet destination is 3 hops to ensure energy saving. Packets are considered for detouring only when they satisfy such constraint. Furthermore, non-minimal path routing routes packets away from congested routers and can enable further DVFS. In these cases, saving in static and clock energy is often sufficient to offset the slight increase in dynamic energy.

We define the row-wise packet transversal as the x-direction and column-wise transversal as the y-direction. When a packet is injected into the network and processed by a router, the router decodes the packet and obtain its slack information, and adopt different routing options to the packet based on its slack: minimal path routing is applied for packets with a slack of 0; and non-minimal path routing algorithm is performed on packets that satisfy the distance constrains. The algorithm eventually selects a neighbor with the lowest router utilization and forwards the packet. If the neighbor router happens to be a candidate from minimal paths, the packet slack level remains unchanged; otherwise, slack level is deducted by one. Details of the routing algorithm are described in Algorithm 1. Notice that the algorithm itself is not deadlock-free, so methodologies from [7] is used to avoid deadlock. In other words, the base level of VC is designed to be deadlock-free [5]. If the lowest utilized neighbor router is likely to create a deadlock, the packet will be routed over the base level VC.

Algorithm 1 Non-minimal adaptive routing for GPU packets

```
Require: Current router coordinate, $(C_x, C_y)$
  Destination router coordinate, $(D_x, D_y)$
  In-coming direction of the packet, $Dir_{in}$
  Packet slack level, $SL$
Ensure: Out-going direction of the packet, $Dir_{out}$
  $Avail\_Dir\_set \leftarrow \{\}$;
  $Min\_Path\_set \leftarrow \{\}$;
  $e_x \leftarrow |C_x - D_x|$;
  $e_y \leftarrow |C_y - D_y|$;
  if ($e_x = 0$ and $e_y = 0$) then
    Route the packet to local node and exit;
  else
    $Avail\_Dir\_set \leftarrow minimal\_routes()$;
    $Min\_Path\_set \leftarrow minimal\_routes()$;
    $//minimal\_routes()$ returns directions nearer to the packet's destination
    if ($SL > 0$) then
      if ($e_x \geq 3$) then
        $Avail\_Dir\_set = Avail\_Dir\_set \cup \{North, South\}$;
      end if
      if ($e_y \geq 3$) then
        $Avail\_Dir\_set = Avail\_Dir\_set \cup \{East, West\}$;
      end if
    end if
    $Avail\_Dir\_set = \{Avail\_Dir\_set - Dir_{in}\}$;
  end if
  Select a direction $Dir$ with minimal utilization
  from $Avail\_Dir\_set$ to forward the packet;
  if ($Dir_{out} \notin Min\_Path\_set$) then
    $SL = SL$-1;
  end if
```

4. EVALUATION INFRASTRUCTURE

We build a heterogeneous multicore simulator that integrates both CPUs and GPUs. CPU and the memory hierarchy is modeled after the Simics-based [14] GEMS [15] simulator. And the GPU cores are modeled after GPGPU-Sim [3]. The interconnection network and its power model is based on Garnet [2] and Orion 2.0 [10], respectively.

In this work, we simulate a 36-tile heterogeneous system with 65nm technology nodes. The architecture parameters can be found in Table 1. Tiles are configured to be comparable in size, and each tile contains approximately 100 million transistors.

The optimal topology of heterogeneous multi-core systems remains an open question in the computer architecture community. In this work we choose tile-based architecture since it provides scalability for managing design complexity and can effectively utilize the on-chip resources. Figure 6 shows a 36-tile system connected with routers to a 6x6 mesh network. A tile consisting of a CPU core as well as an L1 cache is denoted by *C*. A tile that consists of a bank of shared L2 cache is denoted by *L2*, and a tile in which a GPU SM resides is denoted by *G*. Off-chip memories are connected via memory controllers labeled with *M*. Components within each tile are connected with a flexible pipeline router *R*, and on-chip traffic is transmitted via routers and links. The rationale for the topology lies in that if we consider four tiles as a cluster (circled with dashed line), except for clusters with memory controllers providing off-chip access, each cluster contains one CPU computation core, one L2 cache and two co-processors, namely SM cores. This structure can be easily extended to a larger system with more tiles, and enables parallelism in finer granularity. Instead of providing CPU and GPU cores with separate networks, sharing the same network guarantees reasonable on-chip resource sharing. An alternative would be all of the components within a cluster sharing a single router, resulting in fewer routers. However, network bandwidth becomes a bottleneck in this design. Although wider links can be provided to guarantee the bi-section bandwidth, energy overhead increases quadratically with the link width.

Restricted by the network size and topology, multiple detours in a single non-minimal path do not provide additional benefits in the evaluated network. Thus, we only consider non-minimal path with a single detour. However, our algorithm can be extended to multiple detours in a larger system with a different topology.

Table 1: Baseline Simulation Configuration

Processor Core	Four-way out-of-order, 6 integer FUs, 4 floating FUs, 128-entry ROB
Private L1 Cache	Split private I/D caches, each 64KB, 2-way set associative, 64B block size, 1-cycle access latency
Shared L2 Cache	16M banked, shared distributed, 4-way set associative, 64B block size, 8-cycle access latency
SM Core	32 width SIMD pipeline, 1024 threads, 8CTA, 16KB shared memory, Round Robin warp scheduling
Memory	4GB DRAM, 200 cycle access latency, 4 memory controllers
Router	5 Input/output ports, 4-stage Pipeline, 6 VCs/port, wormhole routing, 64-bit flits, 4-flit buffer depth, 1 flit per control packet, 9 flits per data packet

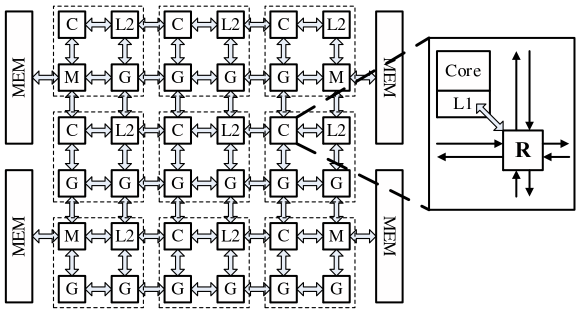
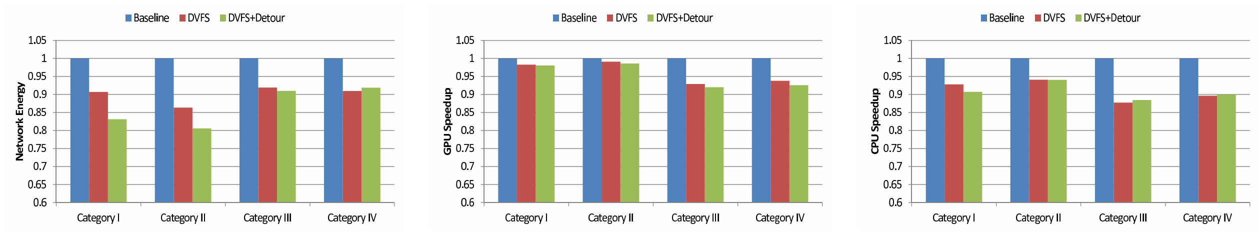


Figure 6: Baseline heterogeneous multi-core system

4.1 Workloads

We use CPU applications from the SPEC OMP 2001 [1] and Nu-MineBench [19] benchmark suites and GPU applications from [3] for our evaluation. We consider 6 CPU benchmarks and 5 GPU benchmarks, where CPU benchmarks include: AFI, AMMP, ART, EQUAKE, KMEANS, and SCALPARC; and GPU benchmarks include: BLACKSCHOLES, LPS, LIB, NN and BFS. The CPU benchmarks are grouped into memory-bound (ART, AMMP, and SCALPARC) and computation-bound (EQUAKE, KMEANS, and AFI) based on their L1 cache miss rate.



(a) Network energy (b) GPU Application Performance (c) CPU Application Performance

Figure 7: Comparison of energy efficiency and performance impact for all application mix categories. *Baseline* corresponds to network using canonical four-stage-pipeline routers without DFVS, minimal path adaptive routing. *DVFS* corresponds to network using flexible-pipeline routers with DVFS technique deployed, minimal path adaptive routing. *DVFS+Detour* corresponds to network using flexible-pipeline routers with DVFS technique deployed, non-minimal path adaptive routing. *Speedup* is defined as cycles per instruction. All results are normalized to *Baseline*.

GPU benchmarks are grouped into latency-tolerant (BLACKSCHOLES and LPS) and latency-intolerant (LIB, NN, and BFS) based on the amount of slack they are able to tolerate. Heterogeneous GPU-CPU workload is created by executing multiple copies of the CPU benchmakrs on multiple CPU cores and one GPU kernel across all SM cores. In each simulation, we sample a period of execution that corresponds to one billion CPU instructions. We enumerate all possible combinations of the CPU and GPU workloads and evaluate 30 workload mixes. The results are presented in four categories: (I) memory-bound CPU and latency-tolerant GPU benchmarks; (II) computation-bound CPU and latency-tolerant GPU benchmarks; (III) memory-bound CPU and latency-intolerant GPU benchmarks; and (IV) computation-bound CPU and latency-intolerant GPU benchmarks.

5. EXPERIMENTAL RESULTS

In this section, we evaluate a heterogeneous multicore system as described in Section 4, with flexible-pipeline routers [23] as described in Section 3.1. We extend the system with DVFS support (Section 3.2) and non-minimal path routing (Section 3.3), and evaluate their impact on NoC energy consumption as well as overall system performance.

Figure 7(a) shows that the impact of DVFS and non-minimal path routing on NoC energy consumption depends on the workload mix. The impact is most significant for workload mix in Categories I and II: compared to the *Baseline*, DVFS can reduce network energy consumption by up to 14%; incorporating non-minimal path routing further reduces energy consumption by 7%. It is worth pointing out that workload mixes in Categories I and II demonstrate additional energy saving from non-minimal path routing, while workload mixes for Categories III and IV do not. This is because GPU workloads in Categories III and IV cannot tolarate slack, and thus are unable to benefit from non-minimal path routing; whereas GPU workloads in Categories I and II can tolerate slack and are able to benefit. Although non-minimal path routing penalizes GPU packets' transmission latency, its performance impact is minimal, as shown in Figure 7(b). However, DVFS does degrade GPU performance for workload mixes in Categories III and IV. This is because GPU workloads in these mixes cannot tolarate latency, and reducing NoC operating frequency increases congestion and latencies. It is possible to change energy/performance trade-off using different DVFS triggering thresholds.

DVFS degrades CPU application performance, as shown in Figure 7(c), mainly due to the increase in network congestion. Although flexible-pipeline routers avoid packet transmission delay in the absence of network congestion, they cannot eliminate congestion. Non-minimal path routing have minimal impact on CPU performance, with the exception of Category I workload mixes. This performance penalty is introduced by congestion caused by detoured packets. Prioritizing CPU packet can potentially eliminate this impact. However, since this performance impact is relatively small, packet prioritization is not evaluated in this paper.

Among all four categories, the second category, namely, the mix of computation-bound CPU and latency-tolerant GPU applications, suffers from the least performance penalty whereas energy savings is the most significant. This is because of the inherent low utilization in some of the routers for light memory-bound CPU applications. Even if GPU packets are detoured via these routers, there is less likelihood that router utilization exceeds the threshold that requires a boost in frequency. In this sense, low utilized routers remain low in utilization while busy routers are released, leading to more energy reduction. To sum up, DVFS becomes the potential performance bottleneck in heterogeneous systems with significant throughput requirement, while non-minimal path routing can be selectively adopted to achieve better energy efficiency.

Table 2: Percentage of detoured packets

BlackScholes	LPS	BFS	NN	LIB
15.193%	5.287%	0.853%	0.002%	0.000%

Table 2 shows the percentage of the packets that are detoured in different applications averaged across all categories. Both BLACKSCHOLES and LPS, applications we classified into the latency-tolerant category, show higher percentage of detoured packets. This result is consistent with our methodology of exploiting GPU slack suggested in Section 2. The number of packets that are routed with non-minimal paths is proportional to the number of packets that have slack. Allowing more packets to be sent through non-minimal paths guarantees a higher possibility of balancing the network traffic, which potentially enables network frequency scaling.

6. RELATED WORK

DVFS has been widely investigated to allow the network to operate at various frequencies. Previous work in [11,12,21] has specifically focused on link latency/power. Mishra *et al.* [17] enabled DVFS on NoC routers to reduce the network power consumption. Flexible-pipeline [23] and variable-pipeline [8,16] routers are also proposed to save network energy while mitigating the latency penalties brought by network frequency reduction. Our work differs from previous work in that, we apply the DVFS technique in addition to flexible-pipeline routers, allowing aggressive voltage/frequency scaling without causing significant performance degradation.

The concept of CPU packet slack was presented in [6], and they proposed an arbitration mechanism for system performance optimization by predicting packet slack. Our work differs from [6] in that we predict the slack for GPU packets and use the predicted slack information for energy optimization in heterogeneous multi-core systems.

Adaptive and multi-path routing have been well studied. [22] introduced a global adaptive load-balanced routing algorithm, in which the adaptive decisions are based on the length of injection queues. [9] and [13] proposed dynamic routing algorithms that judiciously switch between deterministic and adaptive routing schemes. Both software [4] and hardware [18] multi-path routing approaches have been proposed, enabling full exploitation of on-chip interconnection resources.

7. CONCLUSION

The energy consumption associated with data communication in an NoC can correspond to a significant portion of system energy in future heterogeneous multi-core processors. In particular, with large input/output buffers, static router energy consumption can be significant when operating at a high voltage and frequency. Diverting heavy traffic from busy routers and amortizing it over the entire NoC enables these routers to operate at a lower frequency with DVFS support, and in turn reduce overall energy consumption. In this paper, we have explored this opportunity with traffic generated by cores that tolerate memory latencies through context switching between threads, such as GPU cores. Our results show that in a tile-based heterogeneous multi-core system containing both CPU and GPU cores, it is possible to reduce NoC energy consumption by up to 14% with DVFS and adaptive routing. Routing traffic that is not sensitive to network latency to a longer and slower route can avoid busy routers at the cost of higher network latency. Supporting non-minimal path routing enables more routers to operate at a lower voltage and frequency, and thus further lower NoC energy consumption. In particular, for GPU applications with larger packet slack, we achieve up to 7% additional energy saving. Compared to DVFS, detouring packets in NoCs has a minimal impact on system performance as measured by the speedup of the cores.

Acknowledgment

This work is supported in part by grants from National Science Foundation under CNS-0834599, CSR-0834599, CPS-0931931, and CCF-0903427, a contract from Semiconductor Research Cooperation under SRC-2008-TJ-1819. We would like to thank all anonymous reviewers for their constructive comments that help improve the quality of this paper.

8. REFERENCES

[1] SPEC OMP2001. Available at `http://www.spec.org/omp/`.

[2] Agarwal, N., Peh, L.-S., and Jha, N. Garnet: A Detailed Interconnection Network Model inside a Full-system Simulation Framework. Tech. Rep. CE-P08-001, Princeton University, 2008.

[3] Bakhoda, A., Yuan, G., Fung, W., Wong, H., and Aamodt, T. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software(ISPASS)* (2009), pp. 163–174.

[4] Chen, G., Li, F., and Kandemir, M. Compiler-directed Channel Allocation for Saving Power in On-chip Networks. In *Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages(POPL)* (2006), pp. 194–205.

[5] Chiu, G.-M. The Odd-Even Turn Model for Adaptive Routing. *IEEE Transactions on Parallel and Distributed Systems 11*, 7 (2000), 729–738.

[6] Das, R., Mutlu, O., Moscibroda, T., and Das, C. R. Aergia: Exploiting Packet Latency Slack in On-chip Networks. In *Proceedings of the 37th annual international symposium on Computer architecture(ISCA)* (2010), pp. 106–116.

[7] Duato, J. A New Theory of Deadlock-free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems 4*, 12 (1993), 1320–1331.

[8] Hirata, Y., Matsutani, H., Koibuchi, M., and Amano, H. A Variable-pipeline On-chip Router Optimized to Traffic Pattern. In *Proceedings of the Third International Workshop on Network on Chip Architectures(NoCArc)* (2010), pp. 57–62.

[9] Hu, J., and Marculescu, R. DyAD - Smart Routing for Networks-on-Chip. In *Proceedings of the 41st Design Automation Conference(DAC)* (2004), pp. 260–263.

[10] Kahng, A., Samadi, K., Li, B., and Peh, L.-S. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe(DATE)* (2009), pp. 423–428.

[11] Kim, E. J., Yum, K. H., Link, G. M., Vijaykrishnan, N., Kandemir, M., Irwin, M. J., Yousif, M., and Das, C. R. Energy Optimization Techniques in Cluster Interconnects. In *Proceedings of the 2003 international symposium on Low power electronics and design(ISLPED)* (2003), pp. 459–464.

[12] Lee, S. E., and Bagherzadeh, N. A Variable Frequency Link for a Power-aware Network-on-Chip (NoC). *Integration, the VLSI Journal 42*, 4 (2009), 479–485.

[13] Li, M., Zeng, Q.-A., and Jone, W.-B. DyXY - A Proximity Congestion-aware Deadlock-free Dynamic Routing Method for Network-on-Chip. In *Proceedings of the 43rd Design Automation Conference(DAC)* (2006), pp. 849–852.

[14] Magnusson, P., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., and Werner, B. Simics: A Full System Simulation Platform. *Computer 35*, 2 (2002), 50–58.

[15] Martin, M. M. K., Sorin, D. J., Beckmann, B. M., Marty, M. R., Xu, M., Alameldeen, A. R., Moore, K. E., Hill, M. D., and Wood, D. A. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *SIGARCH Computer Architecture News 33*, 4 (2005), 92–99.

[16] Matsutani, H., Hirata, Y., Koibuchi, M., Usami, K., Nakamura, H., and Amano, H. A Multi-Vdd Dynamic Variable-Pipeline On-Chip Router for CMPs. In *Proceedings of the 17th Asia and South Pacific Design Automation Conference(ASP-DAC)* (2012), pp. 407–412.

[17] Mishra, A. K., Das, R., Eachempati, S., Iyer, R., Vijaykrishnan, N., and Das, C. R. A Case for Dynamic Frequency Tuning in On-chip Networks. In *Proceedings of the 42nd annual IEEE/ACM International Symposium on Microarchitecture(MICRO)* (2009), pp. 292–303.

[18] Murali, S., Atienza, D., Benini, L., and De Micheli, G. A Method for Routing Packets Across Multiple Paths in NoCs with In-Order Delivery and Fault-Tolerance Gaurantees. *VLSI DESIGN 2007* (2007), 1–11.

[19] Narayanan, R., Ozisikyilmaz, B., Zambreno, J., Pisharath, J., Memik, G., and Choudhary, A. MineBench: A Benchmark Suite for Data Mining Workloads. In *2006 IEEE International Symposium on Workload Characterization* (2006), pp. 182–188.

[20] Nickolls, J., and Dally, W. The GPU Computing Era. *Micro, IEEE 30*, 2 (2010), 56–69.

[21] Shang, L., Peh, L.-S., and Jha, N. K. Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture(HPCA)* (2003), pp. 91–102.

[22] Singh, A., Dally, W., Towles, B., and Gupta, A. Globally Adaptive Load-Balanced Routing on Tori. *Computer Architecture Letters 3*, 1 (2004).

[23] Zhou, P., Yin, J., Zhai, A., and Sapatnekar, S. S. NoC Frequency Scaling with Flexible-pipeline Routers. In *Proceedings of the 2011 international symposium on Low power electronics and design(ISLPED)* (2011), pp. 403–408.