# NoC-Sprinting: Interconnect for Fine-Grained Sprinting in the Dark Silicon Era

Jia Zhan*, Yuan Xie*, Guangyu Sun‡
*The Pennsylvania State University, {juz145,yuanxie}@cse.psu.edu
‡Peking University, gsun@pku.edu.cn

## ABSTRACT

The rise of utilization wall limits the number of transistors that can be powered on in a single chip and results in a large region of dark silicon. While such phenomenon has led to disruptive innovation in computation, little work has been done for the Network-on-Chip (NoC) design. NoC not only directly influences the overall multi-core performance, but also consumes a significant portion of the total chip power. In this paper, we first reveal challenges and opportunities of designing power-efficient NoC in the dark silicon era. Then we propose *NoC-Sprinting*: based on the workload characteristics, it explores *fine-grained sprinting* that allows a chip to flexibly activate dark cores for instantaneous throughput improvement. In addition, it investigates topological/routing support and thermal-aware floorplanning for the sprinting process. Moreover, it builds an efficient network power-management scheme that can mitigate the dark silicon problems. Experiments on performance, power, and thermal analysis show that NoC-sprinting can provide tremendous speedup, increase sprinting duration, and meanwhile reduce the chip power significantly.

**Categories and Subject Descriptors:** C.2 [Computer-Communication Networks]: Network Architecture and Design

**General Terms:** Performance, Design

**Keywords:** Network-on-Chip, Dark Silicon, Computational Sprinting

## 1 Introduction

The continuation of technology scaling leads to a *utilization wall* challenge [21]: to maintain a constant power envelope, the fraction of a silicon chip that can be operated at full frequency is dropping exponentially with each generation of process technology. Consequently, a large portion of silicon chips will become dark or dim silicon, i.e., either idle or significantly under-clocked. However, most previous work focuses on energy-efficient core/cache design while the impact of on-chip interconnect is neglected. In fact, Network-on-chip (NoC) plays a vital role in message passing and memory access that directly influences the overall performance of many-core processors. Moreover, network components dissipate 10% - 36% of total chip power [8,15,20]. Therefore, how to design the interconnection network is critical to tackle the challenges of multicore scaling in the dark silicon age.

Recently, Raghavan *et al.* [17] proposed *computational sprinting*, in which a chip improves its responsiveness to short-burst of computations through temporarily exceeding its sustainable thermal design power (TDP) budget. All the cores will be operated at the highest frequency/voltage to provide instant throughput during sprinting, and after that the chip must return to the single-core nominal operation to cool down. While such mechanism sheds light upon how "dark" cores can be utilized for transient performance enhancement, it exposes two major design issues: First, the role of interconnect is neglected. NoCs consume a significant portion of chip power when all cores are in sprinting mode. When switching back to the nominal mode, only a single core is active. However, the network routers and links cannot be completely powered down, otherwise a gated-off node would block packet-forwarding and the access of the local but shared resources (e.g., cache and directory). As a result, the ratio of network power over chip power rises substantially and may even lead to higher NoC power than that of the single active core. Second, the mode-switching lacks flexibility and only provides two options: nominal single-core operation and maximum all-core sprinting. Depending on the workload characteristics, an intermediate number of active cores may provide the optimal performance speedup with less power dissipation.

To address these two issues, we propose *fine-grained sprinting*, in which the chip can selectively sprint to any intermediate stages instead of directly activating all the cores in response to short-burst computations. The optimum number of cores to be selected depends on the application characteristics. Scalable applications may opt to a large number of cores that can support highly parallel computation, whereas other applications may mostly consist of sequential programs and would rather execute on a small number of cores. Apparently, *fine-grained sprinting* can flexibly adapt to a variety of workloads. In addition, landing on intermediate sprinting stages can save chip power and slow down the heating process by power-gating the remaining inactive on-chip resources, which is capable of sustaining longer sprint duration for better system performance.

*Fine-grained sprinting* opens up an opportunity to better utilize the on-chip resources for power-efficiency, but it also poses challenges on designing the interconnect backbone. Inherently it incurs three major concerns: *(1) how to form the topology which connects the selected number of cores during sprinting when dark cores and active cores co-exist? (2) how to construct a thermal-aware floorplan of the on-chip resources (cores, caches, routers, etc.) for such sprinting-based multicores? (3) what would be an appropriate NoC power-management scheme?* To answer these questions, we propose a topological sprinting mechanism with deadlock-free routing support, and a fast heuristic floorplanning algorithm to address the thermal problem during sprinting. Moreover, this sprinting scheme naturally enables power gating on network resources in the dark silicon region.

In summary, we propose **NoC-sprinting**, which provides topological/routing support for fine-grained sprinting and employs network power-gating techniques for combating dark silicon. Overall, this paper makes the following contributions:

- Explores challenges and opportunities of designing NoC in the dark silicon era, from the perspectives of both performance and power.
- Investigates the pitfalls of the conventional all-core sprinting which fails to fulfill diverse workload characteristics, and proposes *fine-grained sprinting* for better power-efficiency.
- Proposes NoC support to enable *fine-grained sprinting*, including topology construction, routing, floorplanning, and power management.
- Conducts thermal analysis to evaluate how *NoC-sprinting* correlates with the sprint duration.

Zhan and Xie were supported in part by NSF 0905365 and by the Department of Energy under Award Number DE - SC0005026. Sun was supported by NSFC (No. 61202072), 863 Program of China (No. 2013AA013201), and AMD grant.


*DAC* '14, June 01 - 05 2014, San Francisco, CA, USA

ACM 978-1-4503-2730-5/14/06 ...$15.00
http://dx.doi.org/10.1145/2593069.2593165.

## 2 Challenges and Opportunities

**Dark Silicon and Computational Sprinting.** Conventionally in multi-core scaling, the power gain due to the increase of transistor count and speed can be offset by the scaling of supply voltage and transistor capacitance. However, in today's deep sub-micron technology, leakage power depletes the power budget. We cannot scale threshold voltage without exponentially increasing leakage. Consequently, we have to hold a constant supply voltage, and hence produce a shortfall of energy budget to power a chip at its full frequency. This gap accumulates through each generation and results in an exponential increase of inactive chip resources — Dark Silicon.

Instead of shrinking the chip or sacrificing transistor density, *computational sprinting* [17] embraces dark silicon by leveraging the extra transistors transiently when performance really counts. Special phase change materials should be used as heat storage to support such temporary intense sprinting, leveraging the property that temperature stays constant during the melting phase of the material. Figure 1 demonstrates the nominal single-core operation as well as the sprint mode for a 16-core system. The temperature rises from the ambient environment when the sprint starts at $t_{sprint}$, and then extra thermal energy is absorbed by the melting process of the phase change material, which keeps the temperature at $T_{melt}$. After the material is completely melted, the temperature rises again until $T_{max}$ where the system terminates all but one core ($t_{one}$) to sustain the operation. The system starts to cool after all work is done at $t_{cool}$. Note that numbers in the curve mark different sprint phases and will be analyzed in Section IV.

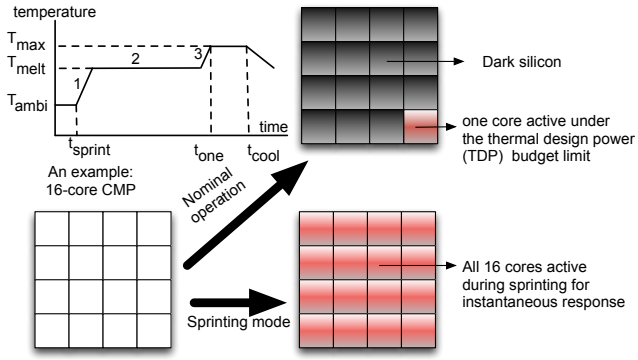



Figure 1: During nominal operation, only one core is active under the TDP constraint, whereas the rest cores are *dark silicon*. In sprinting mode, all the cores will be activated to provide instantaneous throughput.

Conventional computational sprinting still focuses on computation, whereas the role of interconnect is neglected. Here we demonstrate two key challenges that require careful consideration when designing NoC for sprinting-based multicores in the dark silicon age.

**NoC Power Gating.** Power gating is an efficient power-saving technique by completely shutting down the idle cores to reduce leakage. However, as more and more cores turn "dark", the network components such as routers and links also become under-utilized. As mentioned, NoC dissipates 10% - 36% of total chip power [8, 15, 20]; additionally, the more cores become dark, the larger the ratio of network power over the total chip power. This observation also points out the flaw of the conventional computational sprinting which turns off all but one core during nominal operation, while neglecting the impact of NoC.

To give a brief overview of network power, we simulate a classic wormhole router with a network power tool DSENT [19]. The flit width is 128 bits. Each input port of a router comprises two virtual channels (VC) and each VC is 4-flit deep. The power value are estimated with an average injection rate of 0.4 flits/cycle. Figure 2 shows the router power breakdown when varying the operating voltage (1v, 0.9v, 0.75v) and frequency (2GHz, 1.5GHz, 1.0GHz) under 45 nm technology. We can see that leakage power contributes a significant portion to the total network power. In addition, the ratio of leakage power increases as we scale down the supply voltage and frequency, and even exceeds that of dynamic power in some cases.

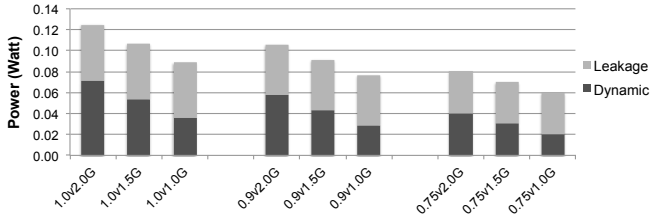



Figure 2: Router power breakdown (dynamic power vs leakage power) when varying the operating voltage and frequency.

Sprinting-based multicores activate a single core during nominal operation whereas the rest are turned off. Figure 3 shows the chip power breakdown when scaling the number of cores based on the Niagara2 [16] processor. We evaluate the power dissipation with McPAT [13] for cores, L2 caches, memory controllers (MC), NoC, and others (PCIe controllers, etc.). We assume idle cores can be gated-off (dark silicon) while *other on-chip resources stay active or idle.*

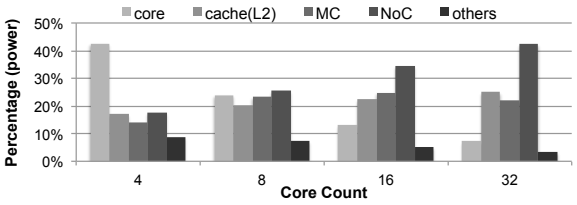



Figure 3: Chip power breakdown during nominal operation (single active core) in sprinting-based multicores. The percentages denote the component power (core, cache, NoC, MC, and others) over the total chip power.

As shown in Figure 3, NoC accounts for 18%, 26%, 35%, and 42% of chip power respectively for 4-core, 8-core, 16-core, 32-core CMP chips when they are operating at nominal mode. In contrast, the power ratio for the single active core keeps decreasing as the "dark silicon" grows. Therefore in this scenario, it is inappropriate to only measure core power when power budget is the design limitation.

NoC power gating is heavily dependent on the traffic. In order to benefit from power gating, an adequate idle period (namely, "break-even time") of routers should be guaranteed to make sure they are not frequently woken up and gated off. Recently researchers have proposed various schemes [4, 5, 14, 18] to mitigate the latency overhead caused by frequent router wake-up. However, these techniques do not account for the underlying core status and will result in sub-optimal power gating decisions.

**Workloads-Dependent Sprinting.** A straightforward sprinting mechanism is to transiently activate all the dark cores at once. However, this scheme fails to explore the sporadic workload parallelism and thus may waste power without sufficient performance gain, especially for multi-threaded applications that have various scalability. Here we use PARSEC 2.1 [2] as an example. We simulate CMP systems using gem5 [3] and observe the performance speedup when varying the core count. For clarity, Figure 4 selects a few results that can represent different workload characteristics.

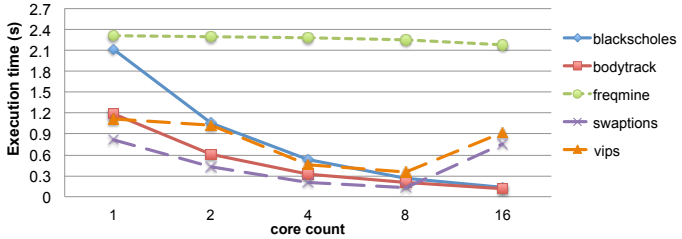



Figure 4: Execution time of running PARSEC benchmarks when increasing the number of available cores.

The detailed evaluation methodology is described in Section IV.

As shown in Figure 4, some benchmarks (e.g. *blackscholes* and *bodytrack*) achieve significant performance speedup as the number of cores increases. In contrast, for *freqmine*, the execution time is almost identical at different configurations, which implies its serial program benefits little from the extra cores. In addition, there are some applications (e.g. *vips* and *swaptions*) that achieve obvious speedup as the core count increases within a small range but then slow down gradually, and further suffer from delay penalty after exceeding a certain number. This is because adding more cores than required by the application parallelism may incur significant overheads that may offset and even hurt performance. The overheads include thread scheduling, synchronization, and long interconnect delay due to the spread of computation resources. Therefore, for sprinting-based multicores, activating all the dark cores is not a universal solution for all applications.

# 3 Our Method: NoC-Sprinting

As illustrated above, an efficient sprinting mechanism should be able to provide different levels of parallelism desired by different applications. Depending on workload characteristics, the optimal number of cores required to provide maximal performance speedup varies. This also raises challenges in designing a high performance and low power interconnect to support the sprinting process.

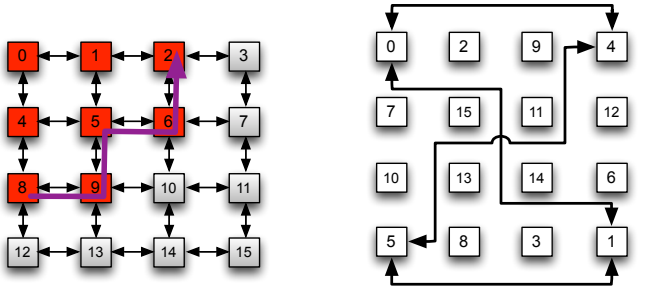
## 3.1 Fine-Grained Sprinting

We first propose *fine-grained sprinting*, a flexible sprint mechanism that can activate a subset of network components to connect a certain number of cores for different workloads.

Specifically, during execution, the CMP system may experience a short burst of computation due to the abrupt fluctuation of a running program. As such, the system will quickly react to such intense computation and determine the optimal number of cores that should be offered for instantaneous responsiveness. Then the system will activate the required number of cores while the others remain "dark". There are some existing work [6, 12] on adapting system configurations like core count/frequency to meet runtime application requirements. *Since our focus is on how to design interconnect under such circumstances, we assume that these application parallelism can be learnt in advance or monitored during run-time execution.*

## 3.2 Irregular Topological Sprinting and Deadlock-Free Routing

Under the nominal operation, only a single core (namely *master core*) remains active. There are different choices of placement for the master core. We list a few examples here, but real implementations should not be limited by these mentioned conditions. Firstly it could be placed in the center of the chip to reduce the transmission latency for thread migration. Another example is to select the core running the OS as the master core since it is always activated. The core next to the memory controller is also a good candidate if the application generates intensive memory accesses. Without loss of generality, we choose the top-left corner node as the master node which is closest to the memory controller, i.e., Node 0 as shown in Figure 5a.




(a) Logical connection of a 16-node mesh network. The irregular topology and convex DOR routing.

(b) Physical allocation for the original network in (a). Only links for 4 nodes are shown for clarity.

Figure 5: Topology, routing, and floorplan for fine-grained sprinting

After the system transfers to the sprinting mode, a number of cores will be activated and keep running for a short duration. *Fine-grained sprinting* requires topological support from the following aspects:

- Pay-as-you-go: fine-grained activation of any number of cores.
- Short communication delay between different nodes, especially to the master node where the memory controller resides.
- Routing should be simple and deadlock-free which does not incur significant control complexity or hardware overhead.

To achieve these goals, we propose to start from the master node, and connect other nodes to the network in ascending order of their Euclidean distances to the master node. For example, the red nodes in Figure 5a demonstrate the topology of a 8-core sprinting. Note that we use Euclidean distances instead of Hamming distances here. The latter may guarantee a shortest routing distance between the newly-added node to the master node, but would generate longer inter-node communication to other nodes. For example, both cases would choose node 0, 1, and 4 as 3-core sprinting. But if 4-core sprinting is triggered, the method with Hamming distance may possibly choose node 2 whereas the method with Euclidean distance would generate a better choice by accommodating node 5. Algorithm 1 generates the order of $N$ nodes used for *topological sprinting*.

**ALGORITHM 1.** Irregular Topological Sprinting

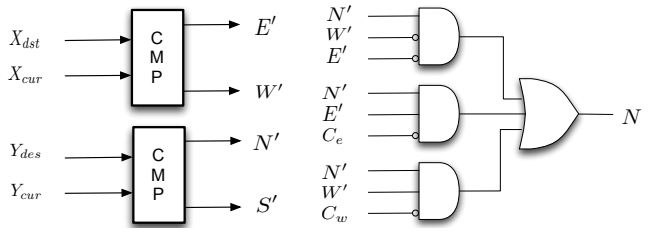
**Result**: A linked-list $L$ of routers to be activated
**Initialize**: $D[i] = 0, i = 0, 1, 2...N-1$. The coordinate for $R_k$ is $(x_k, y_k)$.
**for** $i \leftarrow 1$ **to** $N-1$ **do**
  $D[i] = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2}$;
**end**
Sort $R[i](i = 0, 1...N-1)$ in ascending order of $D[i](i = 0, 1...N-1)$ and put them into a linked-list $L$. Break ties according to the order of indexes.

The *fine-grained sprinting* process will generate an irregular network topology to connect active cores. Meanwhile, it guarantees that chosen nodes would form a convex set in the Euclidean space, i.e., the topology region contains all the line segments connecting any pair of nodes inside it. Flich *et al.* [7] proposed a distributed routing algorithm for irregular NoC topologies but their algorithm requires twelve extra bits per switch. Adapted from their approach, we extend the Dimension-Order-Routing (specifically, X-Y routing) algorithm for such convex topologies (CDOR). Specifically, two connectivity bits ($C_w$ and $C_e$) are leveraged to indicate whether a router is connected to its western or eastern neighbors. As in conventional DOR, we assume that X and Y coordinates of the final destination are stored in the packet header ($X_{des}$ and $Y_{des}$), and each switch knows its X and Y coordinates (through $X_{cur}$ and $Y_{cur}$ registers at each switch). The origin of the coordinate system is located at the top-left corner of the 2D mesh. Messages are routed from the current router to the destination router, according to the offsets of coordinates and the two connectivity bits per router. Figure 5a shows a routing path from the source to its destination. The detailed routing algorithm is described in Algorithm 2. Furthermore, Figure 6 depicts the routing logic design, which includes two comparators per switch and the routing circuit for computing the North port. The routing logic for other ports can be designed similarly based on Algorithm 2. We implemented CDOR on behavioral Verilog. Synthesized results using Synopsys Design Compiler (45nm technology) show that it adds less than 2% area overhead compared to a conventional DOR switch.

**ALGORITHM 2.** Convex Dimension Order Routing (CDOR)

**if** $X_{des} > X_{cur}$ *and* $C_e = 1$ **then**
  output_port = east;
**else if** $X_{des} < X_{cur}$ *and* $C_w = 1$ **then**
  output_port = west;
**else if** $Y_{des} > Y_{cur}$ **then**
  output_port = north;
**else if** $Y_{des} < Y_{cur}$ **then**
  output_port = south;
**else**
  output_pot = local;
**end**

DOR (such as deterministic X-Y routing) is deadlock-free because some of the turns are eliminated such as SE, NW, NE, and SW (S, W,




(a) Two comparitors per switch (b) Routing logic of the North port

Figure 6: Routing logic for convex DOR algorithm

N and E represent south, west, north and east, respectively). In our CDOR, although the NE turn may happen, it is deadlock-free. For example, as shown in Figure 5a, a NE turn happens at Node 5 but this also indicates the east output port of its southern neighbor 9 is not connected. Therefore a WN turn cannot happen and thus eliminate a cycle that may cause a deadlock.

### 3.3 Thermal-Aware Floorplanning

The key design constraint of *fine-grained sprinting* is the thermal design power (TDP). The above topological sprinting process does not consider thermal behavior to simplify control and routing logic. Therefore, here we propose a design-time floorplanning algorithm that can be seamlessly integrated with the topological sprinting process while providing better thermal distribution to avoid hot spots. Thus, it sustains a longer sprint duration by slowing down the heating process.

Consider a 4-core sprinting in a 16-node mesh network as shown in Figure 5a. We may opt to choose the top-left four nodes for better performance, but alternatively may prefer the four scattered corner nodes from the thermal point of view. To overcome this dilemma, *we still maintain the original logic connectivity of the mesh network in consistent of the topological sprinting process, but propose a heuristic algorithm that reallocates the physical location of each node.*

As shown in Algorithm 3, our floorplanning algorithm treats the 2D mesh network as a graph, and allocates the nodes based on the list generated from Algorithm 1. In our annotations, $G$ represents the original logical network, $S$ contains the set of nodes that have already been explored in $G$, $G'$ represents the physical floorplan, and $S'$ is the corresponding set of occupied nodes in $G'$. At each iteration, it picks up a node $R_k$ in $G-S$, and maps it to a node in $G'-S'$ that has the maximum weighted sum of Euclidean distances to all the nodes in $S'$ for the optimal thermal distribution. This process is described in Function *MaxWeightedDistance* as in Algorithm 4. Note that the weight of a distance is inversely proportional to the Hamming distance between $R_k$ and the node in $S$. The rationale behind this scheme is that, the longer the Hamming distance in logical connectivity, the less chance these two nodes would be selected together during sprinting and accumulate heat dissipation, thus they can be placed closer in the physical floorplan.

**ALGORITHM 3.** Thermal-aware heuristic floorplanning algorithm

**Result**: Positions for all nodes
**Initialize**: Original floorplan $f$: {$R_0$, $R_1$ ... $R_{N-1}$}. Transformed floorplan $f'$: {$R'_0$, $R'_1$ ... $R'_{N-1}$}. The coordinate for $R_k$ or $R'_k$ is $(x_k, y_k)$.
Set $S=\phi$, $S'$={$R'_0$,$R'_1$ ... $R'_{N-1}$}. Queue $Q=\phi$. List $L$ from Algorithm 1
**Goal**: Find the mapping $Pos()$ from $f$ to $f'$.
$Pos(R_0)$ = 0(Master Node); Put $R_0$ in $S$; Delete $R'_0$ from $S'$;
Put all unexplored adjacent nodes of $R_0$ into $Q$ based on List $L$;
**while** $Q \neq \phi$ **do**
  $R_k$ = $Q$[0]; Delete $Q$[0] from $Q$;
  $Pos(R_k)$ = **MaxWeightedDistance**$(S, S', R_k)$;
  Delete $R'_{Pos(R_k)}$ from $S'$; Put $R_k$ in $S$;
  Put all unexplored adjacent nodes of $R_k$ into $Q$ based on List $L$;
**end**

The floorplanning algorithm frees the sprinting process and routing algorithm from the thermal concern, i.e., only the logical connectivity of mesh network needs to be considered during topological sprinting. Figure 5b shows the final floorplan of the physical network and only links for four-core sprinting are shown for clarity. Note that the floorplanning algorithm will increase the wiring complexity and generate long links. A standard method of reducing delay of long wires is to insert repeaters in the wire at regular intervals. Recently, Krishna *et al.* [11] have validated such clockless repeated wires that allow multi-hop traversals to be completed in a single clock cycle.

**ALGORITHM 4.** MaxWeightedDistance$(S, S', R_k)$

**Initialize**: $Sum = 0$; $Max = 0$;
**for** *every node* $R'_i$ *in* $S'$ **do**
  **for** *every node* $R_j$ *in* $S$ **do**
    $w_{ij} = 1/(|x_k - x_j| + |y_k - y_j|)$;
    $d_{ij} = \sqrt{(x_i - x_{Pos(R_j)})^2 + (y_k - y_{Pos(R_j)})^2}$;
    $s_{ij} = w_{ij} * d_{ij}$;
  **end**
  $Sum = \sum s_{ij}$;
  **if** $Sum > Max$ **then**
    $Max = Sum$; $Pos(R_k) = i$;
  **end**
**end**
Return $Pos(R_k)$;

### 3.4 Network Power Gating

With our proposed *fine-grained sprinting*, the network power gating scheme becomes straightforward. *Since the topological sprinting algorithm activates a subset of routers and links to connect the active cores, we gate off the other network components as shown in the shaded nodes of Figure 5a*. Moreover, the proposed CDOR algorithm routes packets within the active network and thus avoids unnecessary wakeup of intermediate routers for packet forwarding. This further increases the idle period of the dark region for longer power gating.

However, we still need to consider the Last-Level-Cache (LLC) architecture for network power gating. For private per-core LLC, centralized shared LLC, or distributed shared LLC connected with a separate network (NUCA), our power gating mechanism works perfectly without the need for any further hardware support. However, for tile-based multicores where each tile comprises of a shared bank of LLC, there may be some packet accesses to dark nodes for cache resources. Therefore, some complimentary techniques such as bypass paths [4] can be leveraged to avoid completely isolating cache banks from the network. We accommodate this method in our design.

## 4 Architectural Evaluation

We use the gem5 [3] full system simulator to setup a sprinting-based multicore architecture with 16 ALPHA CPUs. We use Garnet [1] to model a $4 \times 4$ mesh network and DSENT [19] for network power analysis. The detailed system configurations are listed in Table 1.

Table 1: System and Interconnect configuration

| | | | |
|---|---|---|---|
| core count/freq. | 16, 2GHz | topology | $4 \times 4$ 2D Mesh |
| L1 I & D cache | private, 64KB | router pipeline | classic five-stage |
| L2 cache | shared & tiled, 4MB | VC count | 4 VCs per port |
| cacheline size | 64B | buffer depth | 4 buffers per VC |
| memory | 1GB DRAM | packet length | 5 flits |
| cache-coherency | MESI protocol | flit length | 16 bytes |

We evaluate *NoC-sprinting* with multi-threaded workloads from PARSEC [2] by assuming the chip can sustain computational sprinting for one second in the worst case, which is consistent with [17]. Later we will analyze how *NoC-sprinting* influences the sprint duration. We first start running the benchmarks in a simple mode and take checkpoints when reaching the parallel portion of the program. Then, the simulation restores from checkpoints and we record the execution time of running a total of one billion instructions for each benchmark. In addition, we construct synthetic traffic for further network analysis.

### 4.1 Performance Evaluation

Here we evaluate how NoC-sprinting improves the system responsiveness. In comparison, one naive baseline design (*non-sprinting*) is to always operate with one core under TDP limit. Another extreme case (*full-sprinting*) is to activate all the 16 cores during sprinting. While the methods to predict the application parallelism [6, 12] is beyond the scope of this paper, we conduct off-line profiling on PARSEC to capture the internal parallelism of the

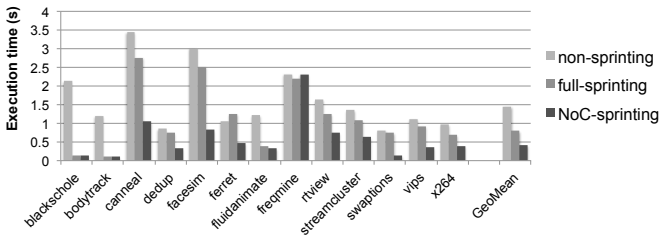



Figure 7: Execution time comparison with different sprint mechanisms.

benchmarks. Figure 7 shows the execution time of PARSEC workloads with different sprinting schemes.

We can see that *NoC-sprinting* cuts down the execution time substantially compared to *non-sprinting*. It achieves 3.6x speedup on average for all the applications. In comparison, *full-sprinting* fails to provide the maximal speedup in some cases with an average 1.9x speedup. It is because increasing the active core count in some programs would incur overheads that may outweigh achievable benefits after a saturating point is reached. These overheads come from OS scheduling, synchronization, and long interconnect delay due to the spread of computation resources.

### 4.2 Core Power Dissipation

Instead of waking up all the dark cores for quick response, *NoC-sprinting* provides better power-efficiency by allocating *just enough* power to support the maximal performance speedup. Since the triggered topology directly determines the number of cores to be powered on, here we first explore its impact on the core power dissipation. Apart from *full-sprinting*, we also compare *NoC-sprinting* with a naive *fine-grained sprinting* scheme which does not employ any power gating techniques, i.e. the system only cares about selecting the optimal number of cores for actual execution and leaves the others idle.

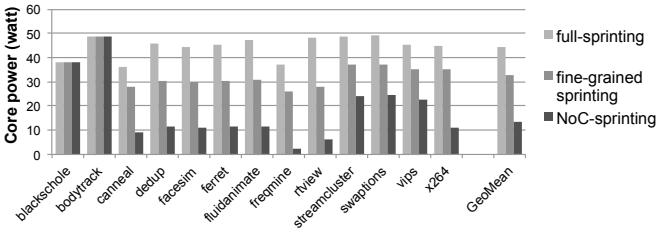



Figure 8: Core power dissipation with different sprinting schemes. Here fine-grained sprinting does not include any power-gating schemes to the idle cores.

As shown in Figure 8, except for *blackscholes* and *bodytrack* which achieve the optimal performance speedup in *full-sprinting* and hence leave no space for power-gating, *NoC-sprinting* cuts down the most power across all the other applications. Compared to *full-sprinting*, *fine-grained sprinting* saves 25.5% power even though power gating is not applied. More promisingly, *NoC-sprinting* achieves 69.1% core power saving on average for all applications.

### 4.3 Analysis of On-Chip Networks

*NoC-sprinting* provides customized topology, routing, floorplanning, and efficient power-gating support for fine-grained sprinting. Therefore in this subsection, we evaluate network performance and power to see how the NoC behaves during the sprinting process.

**Network Latency:** *Full-sprinting* activates the entire network and would possibly lose some performance speedup in the long interconnect. In contrast, *NoC-sprinting* uses a subset of routers to directly connect the active cores, which avoids unnecessary network traversals in the dark nodes with the support of CDOR routing algorithm. As an example, Figure 9 shows the average network latency for running PARSEC with different sprinting schemes. Apparently, *NoC-sprinting* shortens the communication latency for most applications. Overall, it cuts down the network latency by 24.5%.

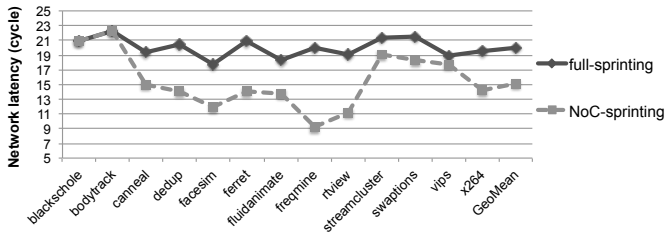



Figure 9: Comparisons of average network latency after running PARSEC with full-sprinting and NoC-sprinting

**Network Power:** As Figure 3 shows, network power becomes more and more significant as cores turn dark. Therefore, optimizing NoC power dissipation becomes an urgent issue in order to combat the power shortage in the dark silicon age.

Figure 10 shows the total network power consumption during the sprint phase of running PARSEC. As we can see, *NoC-sprinting* successfully cuts down the network power if an intermediate level of sprinting is selected. On average, it saves 71.9% power compared to *full-sprinting*. This is because *NoC-sprinting* can adapt the network topology according to workload characteristics and only operates on a subset of nodes. In comparison, *full-sprinting* activates a fully-functional network and loses opportunities for power-gating.

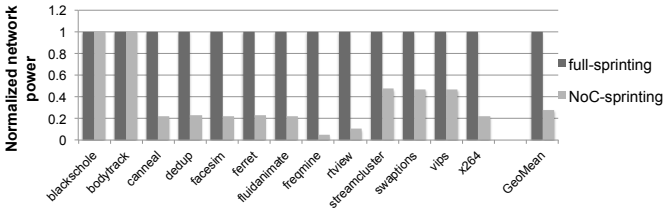


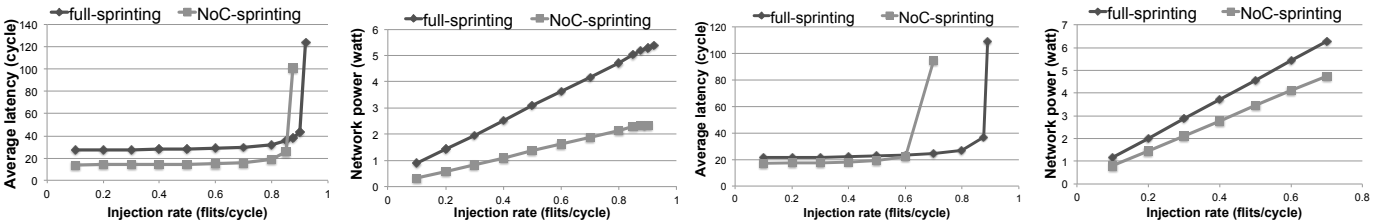

Figure 10: Comparisons of total network power after running PARSEC with full-sprinting and NoC-sprinting

**More Analysis with Synthetic Traffic:** Furthermore, we construct some synthetic traffic on a network simulator booksim 2.0 [10] to test *NoC-sprinting* under different traffic scenarios. For *full-sprinting*, we consider traffic to be *randomly mapped* in the fully-functional network and results are averaged over ten samples. We compare *full-sprinting* with *NoC-sprinting* and observe the differences in performance and power while varying the network load. As an example, Figure 11 shows the results of 4-core and 8-core sprinting for a 16-core system under uniform-random traffic. There are a few key observations:

• As shown in Figure 11a and Figure 11c, *NoC-sprinting* cuts down the average flit latency by 45.1% and 16.1% before saturation for 4-core and 8-core sprinting, respectively, because it uses a dedicated region of network for more efficient communication without traversing the dark region. The latency benefit drops when switching to a higher level of sprinting because less routers/links are wasted as intermediate forwarding stations like *full-sprinting*.

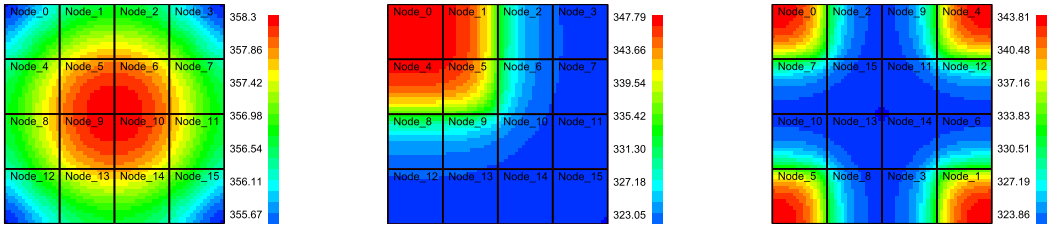
• Correspondingly, *NoC-sprinting* decreases the total network power consumption by 62.1% and 25.9% for 4-core and 8-core sprinting, respectively, as indicated by the gap between the two power curves in both Figure 11b and Figure 11d. The extra routers/links used in *full-sprinting* not only consume leakage power but also generate dynamic power from packet traversals. As expected, the lower sprint level, the more power saving NoC-sprinting can achieve.

• The downside of *NoC-sprinting* is that the network saturates earlier than that of *full-sprinting*. This is because *NoC-sprinting* uses a subset of network where each node is generating and accepting packets. Differently, *full-sprinting* spreads the same amount of traffic among a fixed fully-functional network where some nodes are simply used for intermediate packet forwarding. However, this usually would not affect the network performance in real cases. For example, in the PARSEC benchmarks we have evaluated, the average network injection rate never exceeds 0.3 flits/cycle, which is far from saturating the network.




(a) Latency on 4-core sprint (b) Power on 4-core sprint (c) Latency on 8-core sprint (d) Power on 8-core sprint

Figure 11: Performance and power analysis on full-sprinting and NoC-sprinting with synthetic uniform-random traffic.




(a) Full-sprinting (b) 4-core sprinting with aggregated nodes (c) Thermal-aware NoC-sprinting

Figure 12: Heat maps for running *dedup* with full-sprinting and NoC-sprinting.

### 4.4 Thermal Analysis

*NoC-sprinting* heavily relies on the sprint duration to sustain the desired parallelism. Figure 1 in Section II demonstrated the sprinting process which includes three phases. The duration of each phase is dependent on the property of the phase change material placed close to the die. However, we can still conduct some qualitative analyses to evaluate how NoC-sprinting affects the sprint duration.

Phase 1 indicates that the temperature rises abruptly when sprinting starts, and so does the phase 3 after the melting phase ends. Intuitively, the more power-on components, the faster the temperature will increase. Therefore, *NoC-sprinting* can slow down the heating process by allocating just enough power for the maximum performance speedup. As an example, we analyze *dedup* (one of the PARSEC benchmarks) whose optimal level of sprinting is 4. We collect the power densities using McPAT and feed them into a thermal modeling tool HotSpot [9] as the power trace. As for the floorplan, we abstract the 16-core CMP system as 16 blocks placed in a 2D grid, where each block comprises the Alpha CPU, local caches, and other network resources. We use a fine-grained grid model to observe the stable temperatures of the whole chip. Figure 12 shows the heat maps for *full-sprinting* and *NoC-sprinting*.

As shown in Figure 12a, though power is almost uniformly distributed across the chip, *full-sprinting* results in an overheated spot in the center (358.3°K). In contrast, *fine-grained sprinting* only activates four nodes as shown in Figure 12b and the corresponding peak temperature drops (347.79°K). Furthermore, our thermal-aware floorplanning generates better temperature profile (343.81°K) as shown in Figure 12c.

Phase 2 is the most critical phase that determines the capability of sprinting. Placing phase change materials close to the die elongates this period by increasing the thermal capacitance. Temperature remains stable during melting and the duration of melting is mostly determined by its *latent heat of fusion* — the amount of energy to melt a gram of such material. As such, based on the power results we collected from PARSEC, *NoC-sprinting* increases the duration by 55.4% on average.

As a summary, *NoC-sprinting* reduces the slopes of temperature rise in phase 1 & 3, and enhances the melting duration in phase 2 by slowing down thermal capacitance depletion. Thus, it guarantees a longer sprint for intense parallel computation and further increases the performance.

## 5 Conclusion

In this work, we reveal the challenges and opportunities in designing NoC in the dark silicon age. We present *NoC-sprinting*: it provides topology/routing support, thermal-aware floorplanning, and network power gating for fine-grained sprinting. Our experiments show that *NoC-sprinting* outperforms conventional *full-sprinting* which always activate all the dark cores. It is able to provide tremendous performance speedup, longer sprint duration, and reduces the chip power significantly.

## 6 References


[1] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *ISPASS*, pages 33–42, 2009.

[2] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, 2011.

[3] N. Binkert et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[4] L. Chen and T. M. Pinkston. Nord: Node-router decoupling for effective power-gating of on-chip routers. In *MICRO-45*, pages 270–281, 2012.

[5] R. Das, S. Narayanasamy, S. Satpathy, and R. Dreslinski. Catnap: Energy proportional multiple network-on-chip. In *ISCA*, pages 320–331, 2013.

[6] Y. Ding, M. Kandemir, P. Raghavan, and M. J. Irwin. A helper thread based EDP reduction scheme for adapting application execution in CMPs. In *IPDPS*, pages 1–14, 2008.

[7] J. Flich, S. Rodrigo, and J. Duato. An efficient implementation of distributed routing algorithms for nocs. In *NoCs*, pages 87–96, 2008.

[8] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. A 5-ghz mesh interconnect for a teraflops processor. *Micro, IEEE*, 27(5):51–61, 2007.

[9] W. Huang et al. HotSpot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Trans. on VLSI*, 14(5):501–513, 2006.

[10] N. Jiang et al. A detailed and flexible cycle-accurate network-on-chip simulator. In *ISPASS*, pages 86–96, 2013.

[11] T. Krishna et al. Single-Cycle Multihop Asynchronous Repeated Traversal: A SMART Future for Reconfigurable On-Chip Networks. *Computer*, 46(40):48–55, 2013.

[12] J. Li and J. F. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *HPCA*, pages 77–87, 2006.

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

[14] H. Matsutani, M. Koibuchi, D. Wang, and H. Amano. Run-time power gating of on-chip routers using look-ahead routing. In *ASP-DAC*, pages 55–60, 2008.

[15] T. G. Mattson et al. The 48-core scc processor: the programmer's view. In *SC*, pages 1–11, 2010.

[16] U. G. Nawathe et al. Implementation of an 8-core, 64-thread, power-efficient sparc server on a chip. *IEEE JSSC*, 43(1):6–20, 2008.

[17] A. Raghavan et al. Computational sprinting. In *HPCA*, pages 1–12, 2012.

[18] A. Samih et al. Energy-efficient interconnect via router parking. In *HPCA*, pages 508–519, 2013.

[19] C. Sun et al. DSENT-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *NoCS*, pages 201–210, 2012.

[20] M. B. Taylor et al. The Raw microprocessor: A computational fabric for software circuits and general-purpose programs. *Micro, IEEE*, 22(2):25–35, 2002.

[21] G. Venkatesh et al. Conservation cores: reducing the energy of mature computations. *ACM SIGARCH Computer Architecture News*, 38(1):205–218, 2010.