Network-on-Chip with Long-Range Wireless Links for High-Throughput Scientific Computation

Turbo Majumder, Partha Pratim Pande, Ananth Kalyanaraman
School of Electrical Engineering and Computer Science, Washington State University, Pullman, USA
Email: {tmajumde, pande, ananth}@eecs.wsu.edu

***Abstract*—Several emerging application domains in scientific computing demand high computation throughputs to achieve terascale or higher performance. Dedicated centers hosting scientific computing tools on a few high-end servers could rely on hardware accelerator co-processors that contain multiple lightweight custom cores interconnected through an on-chip network. While network-on-chip (NoC) driven platforms have been studied in the context of accelerating individual applications, this work studies the efficacy of NoC-based platforms to enhance overall computation throughput in the presence of several concurrently executing jobs. Use of long-range links has been shown to reduce network diameter and we use this property in conjunction with different resource allocation strategies to deliver high throughput. Our experiments using a computational biology application suite as a demonstration study show that the use of long-range wireless shortcuts coupled with the appropriate resource allocation strategy delivers computation throughput over 10^{11} operations per second, consuming ~0.5 nJ per operation.**

Keywords-Network-on-chip; long-range links; multicore; computation throughput

I. INTRODUCTION

High-performance scientific computing tools in emerging application domains such as biocomputing demand computation throughputs to scale to terascale and beyond. Given the diversity of tools and the need to cater to a wide user-base, it has become common practice, even within academic settings, to have a dedicated center which hosts a whole range of scientific computing tools on a few high-end data servers. The servers can be expected to service requests from a variety of applications, each with differing resource requirements, and simultaneously support them while delivering high throughput. This server could either be based on a cluster of general purpose microprocessors or make use of a co-processor consisting of a many-core chip where the cores are designed to accelerate targeted operations and are interconnected with an on-chip network. A similar setup is also becoming common practice, albeit on a larger scale, in cloud solution providers (e.g., [1]).

Various studies have shown the efficacy of Network-on-Chip (NoC) driven platforms to accelerate specific

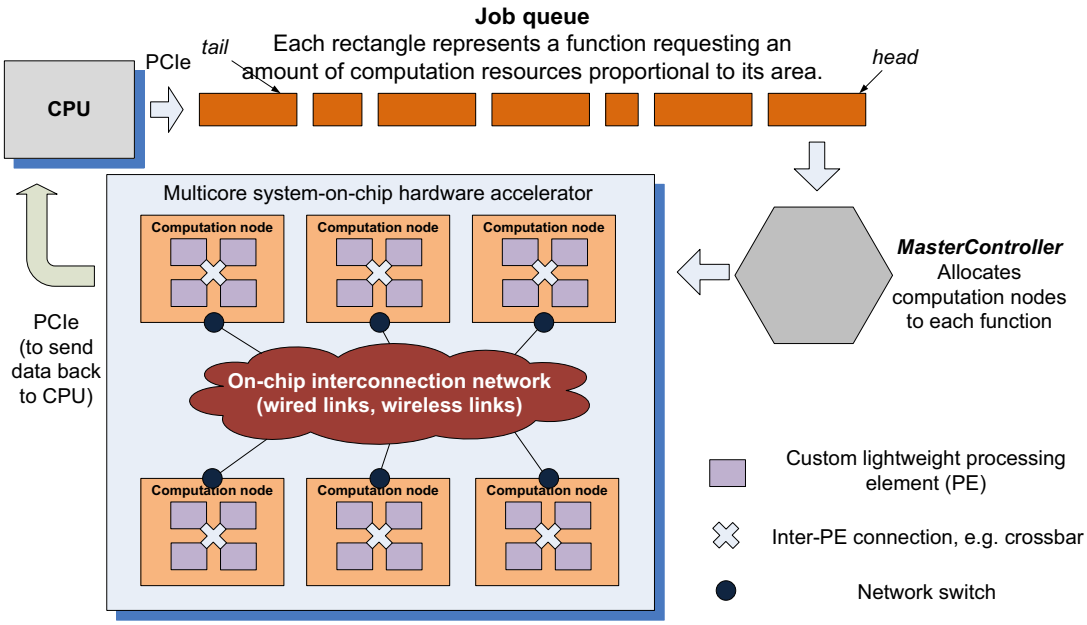

Figure 1. Use-case model of applications requiring high computation throughput

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.72

target applications (e.g., [2]). Although NoC-based platforms have been studied in the context of application throughput, these studies [10][11][12] focus mostly on the network layer, routing and arbitration to improve message throughput. Our focus, in this paper, is on enhancing system-level computation throughput of scientific applications consisting of multiple concurrent jobs with varying footprints. We propose NoC-based platforms to achieve our objective through designing custom processing cores for efficient arithmetic computation, novel task-allocations schemes, and use of on-chip long-range wireless links.

The proposed platforms assume the following *use-case model* (see Fig. 1): A CPU runs the parent process and communicates via an interface (e.g. PCIe) to a multicore system-on-chip that acts as a hardware accelerator for specific computation-heavy kernels. There is a queue of jobs offloaded by the CPU to the hardware accelerator and an allocation unit (*MasterController*) assigns the requested computational resources from the hardware accelerator to the job at the head of the queue. Once some computation resources are assigned to a job, they stay busy till the execution of that particular job concludes, and the result is sent back to the CPU through a similar interface (e.g. PCIe). Each computational resource is a lightweight custom core embedded in a NoC.

The choice of the on-chip network architecture is an important consideration in the design of a NoC-driven platform targeted at enhancing computation throughput. Introduction of long-range links in regular architectures like mesh reduces the overall network diameter and improves inter-core communication latency [3]. It has been shown that the use of on-chip wireless links to implement these shortcuts leads to significant savings in latency and energy, even considering the overhead of wireless transceivers [4]. Our *contribution* here is the design of a NoC-based platform with long-range on-chip wireless shortcuts to enhance the computation throughput of scientific applications with the above-mentioned use-case model.

II. Related Work

The proposed use-case model is relevant for a variety of scientific applications, for example, the use of servers that host application programs to implement standard

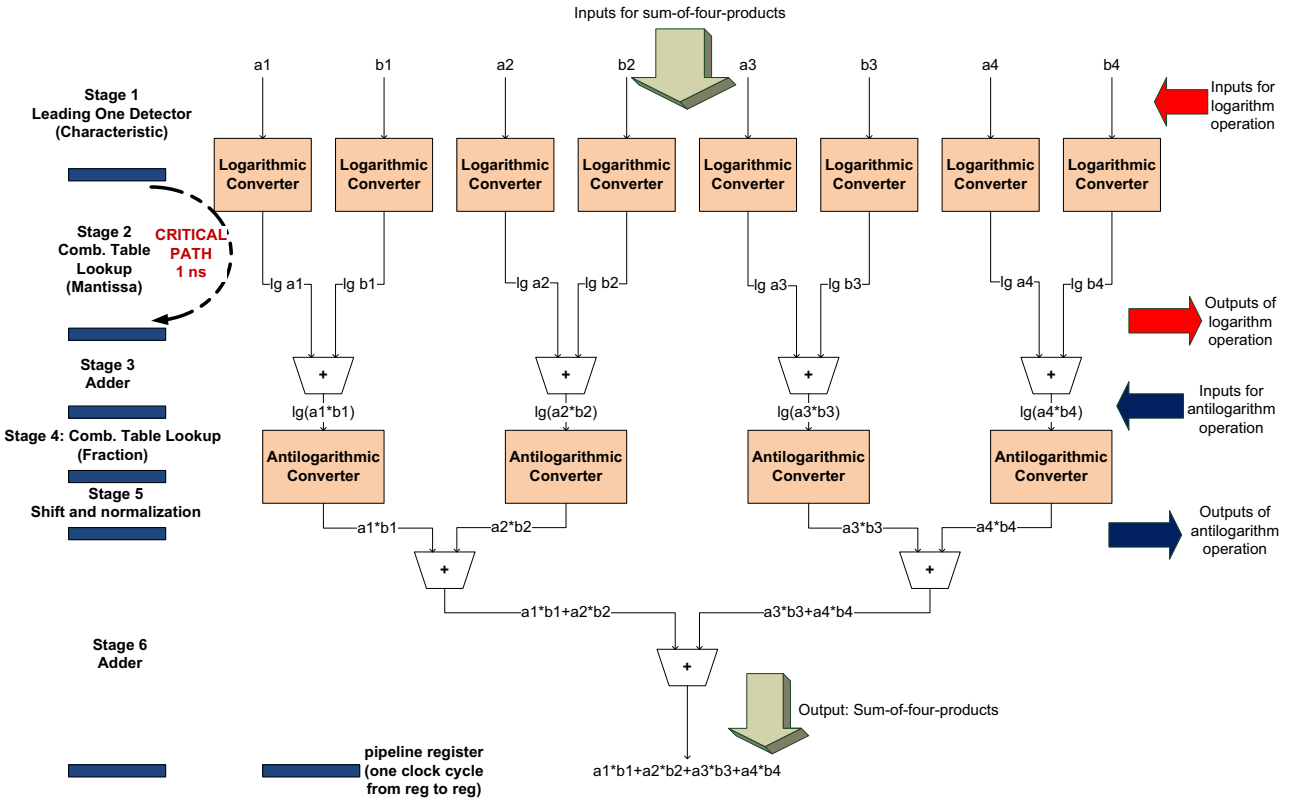

Figure 2. Datapath architecture of each computation core of a processing element (PE)

functions in phylogenetic inference [5], gene sequence alignment [6], climate modeling and weather prediction [7], etc. For instance, a typical genome assembly algorithm farms out billions of pairwise sequence alignment computations, each of which is an independent task that aligns strings of small lengths (e.g., 100-500 base pairs) and can use a small number of cores (e.g., 8-16) [8]. As another example, consider the problem of computing phylogenetic inference using maximum likelihood (ML) [9], where one typically needs to carry out billions of independent tree evaluations, each of which internally performs a small number of floating point calculations using a few cores. There is a considerable body of work in partitioning NoCs for different applications, e.g. [10]. Similarly, there has been work on routing strategies in NoCs that minimize message-level contention between applications, e.g. [11], [12].

NoCs have been shown to perform better by insertion of long range wired links following principles of small-world graphs [3]. Despite significant performance gains, the above scheme implements the long-range links with conventional wires. It has been already shown that beyond a certain length, wireless links are more energy efficient than conventional metal wires. Hence, the performance improvement by using long-range wireless links will be more than that using wired links. Designs of small-world based hierarchical wireless NoC architectures introduced and elaborated in [4] demonstrate this. Our approach leverages the benefits of using long-range wireless shortcuts on a NoC to achieve high-throughput computation for applications belonging to the use-case model considered in this work.

III. Design of NoC with Long-range Links

We present the design of a multicore system-on-chip, where the cores consist of lightweight custom-designed processing elements (PEs), and the on-chip network is a folded torus (network choice explained later). We insert long-range shortcuts using on-chip wireless links on top of the folded torus, and explore different strategies to allocate the computational resources of the system to the application. The details of the system design, wireless shortcut placement, resource allocation and routing are described in this section.

A. Processing Element (PE)

Scientific applications need to carry out a wide range of operations (e.g., log, exp, multiplication, integer comparison, trigonometric functions, etc.). For the sake of design and evaluation in this paper and without loss of generality, we built a PE that performs only a subset of these calculations (using phylogenetic inference as a demonstration study). The system design methodology remains the same if the PE design is modified to implement other functions. We designed a homogeneous multicore system, where a PE computes vector products on floating point numbers and elementary functions like logarithms and exponentials. A PE is one computation node and communicates with other PEs (computation nodes) through the respective network switches and the on-chip network. The number of such nodes represents the system size, *N*, of the multicore system.

We use Fixed-Point Hybrid Number System (FXP-HNS) [13], an efficient and accurate number system to represent floating point numbers. We use 64-bits for number representation; as such our core datapath is 64-bit wide. The architecture of the datapath of a computation core within each PE is shown in Fig. 2. The datapath consists of six pipeline stages, with the functions of each stage indicated in the figure. We consider logarithm and exponential as *basic operations* that the PE is designed to accelerate, while vector product is a *compound operation* involving the basic operations. Logarithm and exponential operations are based on piecewise-linear table-based approximations described in [13], and implemented with logic circuits. In addition, each PE has 2 MB memory in the form of register banks to store inputs and computation outputs. We used Verilog HDL to design the PE along with a wrapper for instruction decoding, data fetching and data write-back. We synthesized the design with 65 nm standard cell libraries from CMP [14]. We determined the clock frequency of 1 GHz based on the critical path (Stage 2) shown in Fig. 2.

B. Network Architecture

Our target is the class of applications that spawn a stream of independent jobs (constituent functions) that individually require variable amounts of computation resources. Communication is necessitated only among nodes catering to a single job during its execution. The location of these nodes on the network can be arbitrary, although preserving locality of allocation becomes important in the interest of keeping the communication overhead low. Given this setup, distributed network architectures such as a Folded Torus are well suited to cater to such traffic patterns. From the VLSI implementation perspective, a torus is a scalable network architecture whose regularity provides for easier timing closure and reduces dependence on interconnect scalability [15]. All inter-node links in the folded torus are one-hop links with respect to the 1 GHz clock used. As mentioned above, this clock frequency requirement arises from the critical path constraint of the PE. Since

our datapath is 64-bit wide, and messages exchanged between nodes contain 64-bit FXP-HNS numbers along with control and routing information, we split each inter-node message into three 64-bit flits – header, body and tail. As a result, each inter-node link needs a minimum bandwidth of 64 Gbps.

C. *Long-Range On-Chip Wireless Links*

From the perspective of the application, we try to minimize the average distances among nodes catering to one job. However, as we explain further in Section III-D, the nodes allocated to a job could turn out to be physically separated on the network, leading to a large communication overhead. From the network architecture point of view, bridging these gaps is possible through the use of long-range point-to-point shortcuts.

Introduction of shortcuts on regular architectures have been shown to provide significant improvements in latency and network throughput for different kinds of applications [3]. Long wired shortcuts however cannot guarantee one-hop transmission and consume significant amounts of energy. Use of on-chip wireless shortcuts overcomes these drawbacks [4].

1) *Physical Layer*

Suitable on-chip antennas are necessary to establish the wireless links. It has been shown that wireless NoCs designed using carbon nanotube (CNT) antennas can outperform conventional wireline counterparts significantly [4]. Antenna characteristics of CNTs in the THz frequency range have been investigated both theoretically and experimentally [16]. Such nanotube antennas are good candidates for establishing on-chip wireless communication links and are henceforth considered in this work. Using CNT antennas, different frequency channels can be assigned to pairs of communicating source and destination nodes, thus creating a form of frequency division multiplexing. This provides dedicated and non-overlapping channels to a

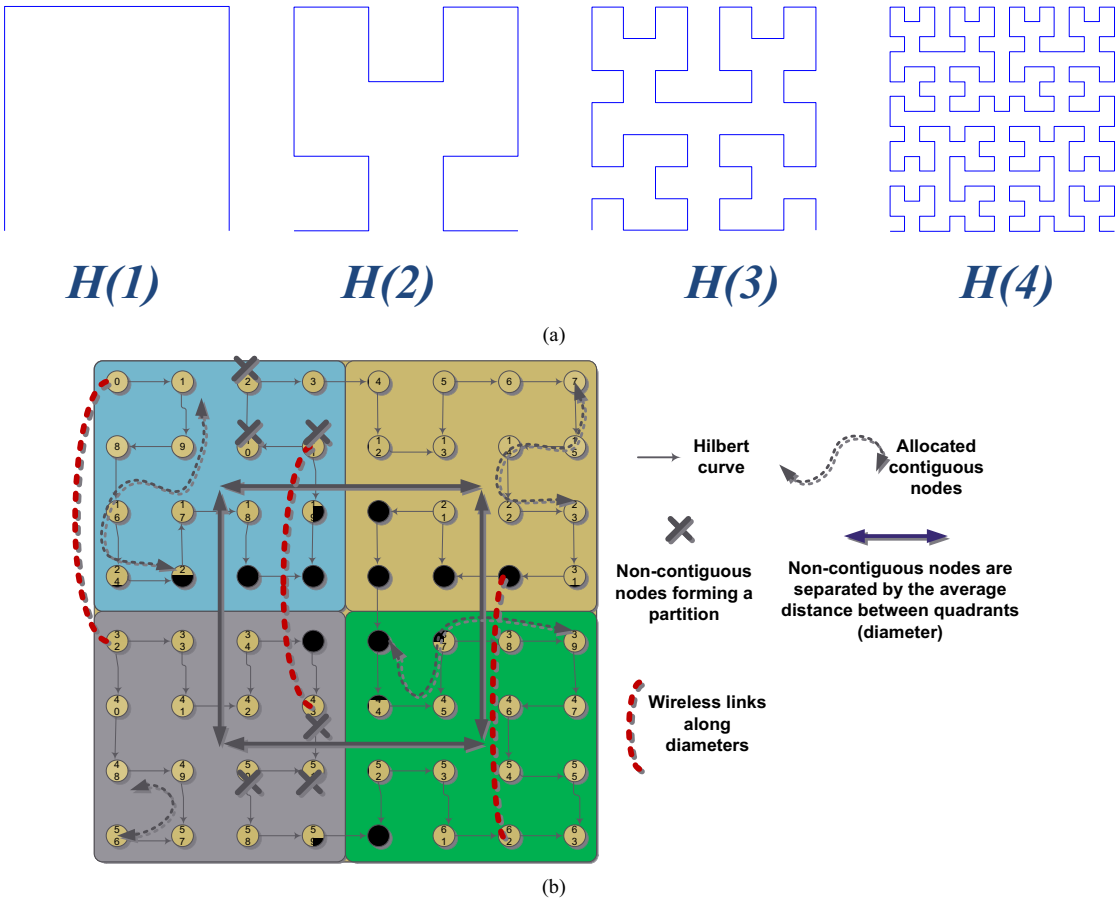

Figure 3. (a) 2-D Hilbert curves with k = 1 (2x2), 2 (4x4), 3 (8x8) and 4 (16x16). For every value of k, there are four Hilbert curves, each rotated by 90 degrees with respect to the next. (b) Each quadrant of a Hilbert curve embedded on the folded torus NoC is shown in a different color. Dynamically allocated partitions can consist of contiguous nodes and non-contiguous nodes. In the latter case, the wireless links act as shortcuts. On an average, the nodes in a non-contiguous partition are separated by a diameter, which justifies wireless link placement along diameters.

source-destination pair. This is possible by using CNTs of different lengths, which are multiples of the wavelengths corresponding to the respective carrier frequencies. It is possible to create 24 non-overlapping wireless channels, each capable of sustaining a data rate of 10 Gbps using CNT antennas, details of which are discussed in [4]. A more detailed description of the physical layer is beyond the scope of this work.

The number of wireless links in our system is determined by the bandwidth each link needs to support. As mentioned earlier, each wireless (inter-node) link needs to sustain a bandwidth of 64 Gbps. Based on the capacity of the wireless channels (10 Gbps), we need 7 channels per link (providing up to 70 Gbps bandwidth). Consequently, the maximum number of single-hop wireless links that we can allow is *int(24/7) = 3*. Note that we could increase the number of wireless links with the same bandwidth if a future technology supports more than 24 non-overlapping channels.

2) Link Placement

The traffic pattern generated by an application determines the most appropriate locations for placement of wireless links. As mentioned earlier, it is not possible to statically predict the traffic pattern. The sets of communicating nodes (executing a job) change with time. Observation of traffic patterns across numerous application maps has shown that among nodes that are not co-located (inter-node hop-count > 2), the probability of pairwise interaction is highest when they are separated by the maximum hop-count along a dimension, or *diameter*. Analytically, this observation can be explained by the fact that the most efficient of the node allocation methods described later in Section III-D divides the network into four quadrants and the need for long-range links arises when allocated nodes are non-contiguous and lie in neighboring quadrants, the mean distance between which is equal to the diameter, as shown in Fig. 3 (b).

Note that we are constrained by only 3 wireless links due to current technology limitations as explained earlier. Again, each application map (in general) would require a different set of specific geometric locations of shortcuts that would bridge large hop-counts. However, we cannot reassign wireless shortcuts for every instance of the application map (each of which takes $\sim 10^{-7}$ s to finish execution, as determined experimentally) as this would lead to an unacceptably large overhead. Hence, we need to determine an optimal placement of these links along torus diameters so that most sets of communicating nodes across all application maps can gainfully access the wireless shortcuts. To this end, we "cover" the entire network by placing them along diameters of the folded torus with *similar angular separation*, as shown in Fig. 3 (b).

D. Dynamic Node Allocation

A network node is *busy* during the execution of a job by the PE; it is *available* otherwise. The computation nodes (PEs) continually send their busy/available status to the allocation unit, *MasterController* (see Fig. 1). When a job requests computation resources, *MasterController* allocates the requisite number of available computation nodes from the system. The nodes thus allocated form a *partition* during the course of function execution and communicate with one another. A desired feature of a partition is that its constituent nodes are co-located so as to minimize the average number of hops spent in message transfers. To this end, an effective allocation strategy would be the one that ensures co-locality without incurring a large allocation time overhead. Simple approaches like breadth-first search do not fit these criteria. We present the following allocation methods, which can be classified into *wireless-agnostic* and *wireless-aware* methods. In the former, the location of wireless links is *not* taken into account *during allocation*, but the links are *used during job execution*. In the latter case, the location of wireless links is taken into account *during* allocation. Additional care is taken to ensure that at most one wireless link is allocated to any given partition.

We also make use of the locality-preserving, space-filling 2-D Hilbert curve [17] for allocation. A Hilbert curve is a recursively constructed space-filling curve where each iteration of the curve contains four copies of the previous iteration rotated so as to align the entry and exit points. Considering discrete iterations of the Hilbert curve, where *H(k)* represents the *k*th iteration, *H(k+1)* is built using four copies of *H(k)*. 2-D Hilbert curves for k = 1, 2, 3 and 4 are shown in Fig. 3 (a). The partitions allocated following the 2-D Hilbert curve are denoted *A-type* if all nodes belonging to that partition are contiguous along wired links on the folded torus; else the partition is *B-type*.

1) Parallel Best-Fit Allocation Using Multiple Hilbert Curves

This allocation strategy preferentially looks for a partition with contiguous nodes to maximize co-locality, and parallelizes the search in order to increase the probability of a quick hit. The algorithm is as follows:

1. First, we use four Hilbert curves on a square folded torus. These four curves are given by three successive right-angle rotations of a single Hilbert curve.

2. We further divide each of the four Hilbert curves into four *segments*, one from each quadrant – thereby resulting in a total of 16 segments (see Fig. 3 (b)). *MasterController* now has 16 heads, each of which is responsible for scanning a segment. All 16 heads act in parallel, to cover different parts of the network simultaneously.
3. Each head now preferentially looks for an A-type partition in its segment. The first head to find such a partition returns it to the requesting job and interrupts all the other scanning heads.
4. In case no A-type partition is found after each head has finished scanning its segment, *MasterController* carries out a serial scan along a Hilbert curve and allocates available nodes as they are encountered.

This method of allocation is wireless-agnostic because we do not make use of the information regarding the location of wireless shortcuts. We refer to the systems using this method as simply *"2D_parallel"* if they do not use wireless shortcuts, and *"2D_parallel + wireless"* if wireless shortcuts are utilized only dynamically during message transfers (i.e., not during allocation) if that reduces the overall distance traversed.

2) *Wireless-First Allocation Using Hilbert Curve*

This is a wireless-aware allocation method in which *MasterController* preferentially looks for available node pairs directly connected by a wireless shortcut. If such a pair is available, they are allocated to the requesting job. *MasterController* then serially scans for the remaining nodes following a Hilbert curve starting from a terminal node of the wireless shortcut. Since only nodes belonging to the same partition communicate with one another, this method ensures that wireless shortcuts are fully utilized. In case no wireless shortcut is available at the time of allocation, nodes are allocated based on a serial scan along the Hilbert curve. We refer to the systems using this allocation method as *"wireless + Hilbert"*.

3) *Wireless-First, Column-Major Allocation*

This is another wireless-aware allocation method, which looks for available wireless shortcuts to be allocated first. The remaining nodes are allocated following the direction of wireless shortcuts such that the nodes in the partition are aligned with the shortcut, so as to maximize the traffic the shortcut carries. As shown in Fig. 3 (b), the wireless shortcuts are placed along the y-axis diameters (columns) of the folded torus. Hence, the node allocation also follows a column-major ordering. The major benefit of this method is that a wireless shortcut can potentially carry traffic from partitions that do not directly include it but are closely aligned with it. Systems using this allocation method are referred to as *"wireless + column-major"*.

4) *Randomized Allocation*

We also explore the simple randomized allocation approach, where *MasterController* maintains a list of available nodes in a random order, and allocates the requested number of nodes from the head of the list. This method of allocation is neither wireless-aware nor does it attempt to achieve any co-locality among the allocated nodes. The only advantage of this allocation method is the simplicity of *MasterController* logic and fewer cycles spent in allocation.

E. *On-Chip Routing*

As mentioned earlier, we adopt wormhole routing to exchange three-flit messages among nodes of a partition. Network switches are based on the designs presented in [18]. Each switch consists of four bidirectional ports (E, W, N, S) to neighboring switches and one local port to/from the computational node. Each port has a buffer depth of two flits and each physical channel is split into 4 virtual channels. We use deadlock-free e-cube routing in torus [19].

For routing in the presence of wireless shortcuts, we need information about the wireless links closest to a source-destination pair, and the bandwidth provided by such links. This information is known beforehand in the form of an *N*-sized routing table available to each router. Based on this knowledge, the router chooses a path via a wireless shortcut if that entails fewer hops to transfer a message between a source-destination pair. The message follows deadlock-free south-last routing [3] when involving wireless shortcuts, and e-cube routing when following wired-only paths between a source and a destination.

IV. EXPERIMENTAL RESULTS

A. *Experimental Setup*

The computation core has a datapath width of 64 bits and provides a number representation accuracy of $\sim 10^{-15}$. Our design could potentially accommodate multiple cores per PE. For the purpose of experiments, we implemented each PE with four computation cores. Each core of the PE has 0.5 MB of register-bank memory associated with it. We synthesized Verilog RTLs for the PEs, the network switches and *MasterController* with 65 nm standard cell libraries from CMP [14]. We used a clock period of 1 ns constrained by the critical path occurring in the core datapath as mentioned in Section III-A. We verified that our design meets all timing constraints, and evaluated power consumption. We laid out the wired NoC

interconnects and determined their physical parameters (power dissipation, delay) using the extracted parasitics (resistances and capacitances). We verified that all wired links could be traversed within one clock cycle. Each wireless link consists of seven channels of 10 Gbps each, providing a total link bandwidth of 70 Gbps. For the wireless links, we considered an energy dissipation of 0.33 pJ/bit as mentioned in [4] to include the energy consumed in the transceiver circuitry and the antennas, and used these to evaluate the total energy consumption of our system.

We experimented with the allocation methods mentioned in Section III-D. We used system sizes of N=64 and N=256 in our experiments. We model the NoC-based multicore platform as a co-processor connected using a PCIe interface. We modeled a PCI Express 2.0 interface using Synopsys ™ Designware ™ IP PCI Express 2.0 PHY implemented on 65 nm process and operating at 5.0 Gbps. We use a 32-lane PCIe 2.0 for our simulation.

We selected a Maximum Likelihood-based phylogenetic reconstruction software called RAxML version 7.0.4 [20], [21] for the purpose of this experimental study. A detailed profiling of RAxML runs using the GNU *gprof* utility reveals that a small set of functions consume a predominant portion (>85%) of the runtime. These functions are denoted by *f6*, *f3* and *f2* respectively based on the computation resources (number of computation nodes) they need for execution. Based on the composition of jobs executing on our system, we bin the system job loads into two categories – one in which *f6* jobs are dominant and the other in which *f3* and *f2* jobs occupy up to half of all the nodes. The total number of jobs concurrently executing on the system is clearly higher in the latter case. Since each *f6* individually requires the largest number of computation nodes (six), the probability that one will be allocated a contiguous partition on the network is relatively low. Therefore, the above test plan represents the conservative end of the spectrum for performance evaluation.

B. *Traffic Pattern Analysis*

Traffic generated as a result of inter-node communication varies with resource (node) allocation schemes, and usage of shortcuts. We analyzed the distribution of probability of interaction of any pair of nodes and their hop-count separation. Note that hop-count depends on whether the NoC uses wireless shortcuts or uses only wireline communication. As expected, a randomized node allocation method is the least expensive in terms of number of clock cycles used by *MasterController*. Using this method, we experimentally found that an average of 85.56 cycles are

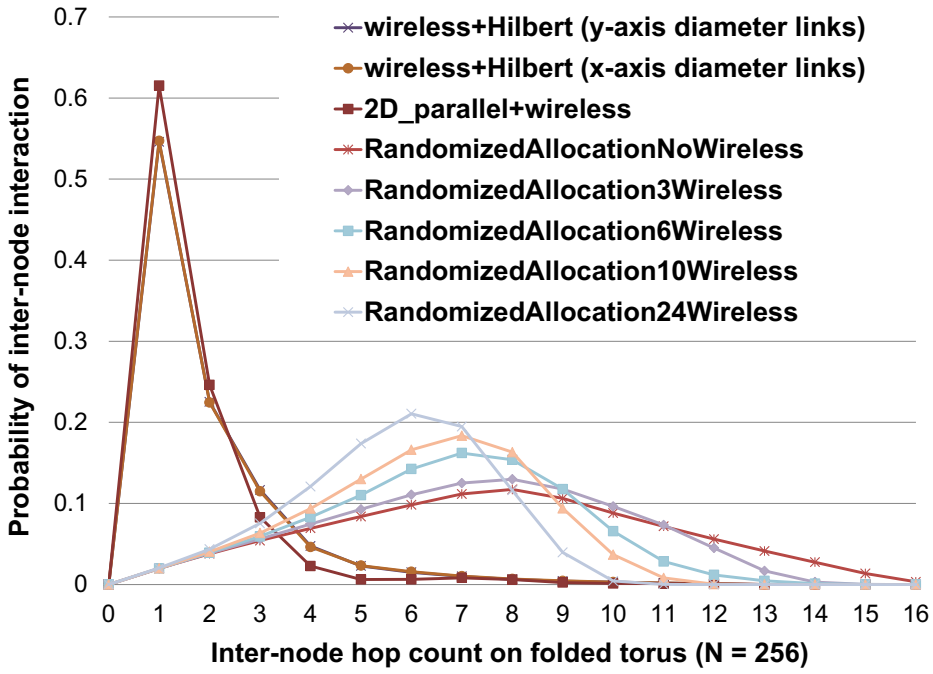

Figure 4. Impact of different node allocation methods and presence of wireless links. For *2D_parallel+wireless* and *wireless+Hilbert*, the average inter-node hop count is mostly below 4, whereas for a randomized allocation method, the average hop-count remains higher even with 24 wireless shortcuts.

spent in allocating 256 nodes to different tasks (each task uses more than one node). On the other hand, allocation methods that preserve co-locality of nodes pertaining to a task – *2D_parallel+wireless* and *wireless+Hilbert* – use an average of 141.54 and 170 allocation cycles respectively. Fig. 4 shows the variation of average probability of interaction between two nodes vs. the hop-count separating them. We plot this for each of the allocation methods described in Section III-D. For *wireless+Hilbert*, we have two different placements of wireless links – along the x-axis and the y-axis diameters. The additional time spent in allocation using *wireless+Hilbert* and *2D_parallel+wireless* is justified, as shown in Fig. 4, because inter-node interaction probability peaks at shorter hop-counts of 1-2 in *2D_parallel+wireless* and *wireless+Hilbert*, whereas randomized allocation peaks are at larger hop-counts between 6 and 8. With addition of 3, 6, 10 or 24 wireless shortcuts, the peak shifts to the left but much less than desired. Consequently, we do not use the randomized allocation method in further experimentation, and use specialized allocation methods that preserve co-locality, as described in Section III-D.

C. Computation Throughput

To measure the computation throughput of our system, we only use each basic operation (log/exp) performed by a core (see Section III-A) as the unit (leaving out addition because it is much simpler), and the number of operations per second as the metric. Computation throughput is not only affected by the mix of jobs running on the system at any point of time, but also by allocation time overhead, usage of wireless shortcuts, and network architecture. Fig. 5 shows the computation throughput for the two different job loads mentioned earlier across different network architectures and system sizes. *2D_parallel + wireless* consistently provides the best computation throughput across job loads and system sizes. It is interesting to note that the best performing architecture has a wireless-agnostic allocation method. While wireless-aware allocation methods guarantee that a larger proportion of flits use the wireless shortcuts (see Fig. 6), this also leads to congestion over these links. Since we use a static routing technique that is based only on the comparison of distances traversed in alternative paths (using shortcuts vs. not using shortcuts), with a wireless-aware allocation strategy, we end up attempting to route more flits through the wireless shortcuts than their bandwidth can sustain without incurring a latency penalty. In a wireless-agnostic allocation method such as *2D_parallel + wireless*, we try to maximize the number of A-type partitions during allocation, leaving to the wireless shortcuts the job of carrying traffic from B-type partitions.

Referring to Fig. 5, we also note that the cases containing a higher proportion of *f2* and *f3* jobs have a 5-10% higher computation throughput than the *f6* dominant loading scenario. Note that a larger system size (Fig. 5(b)) provides proportional gain in computation throughput because the problem size can be appropriately scaled up. The lowest parallelization efficiency is obtained for *wireless + column-major* and this is attributed to the high allocation-time overheads for larger system sizes, proving that this allocation method is less scalable with system size.

D. Proportion of Flits Using Wireless Shortcuts

Fig. 6 shows the percentage of total flits that used the

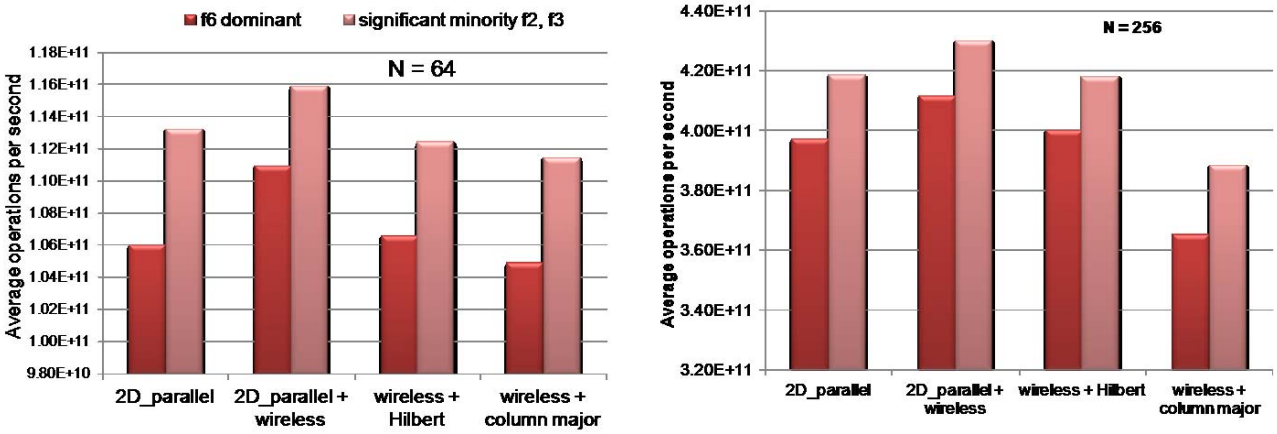

Figure 5. Computation throughput across different network architectures, system sizes and job loads.

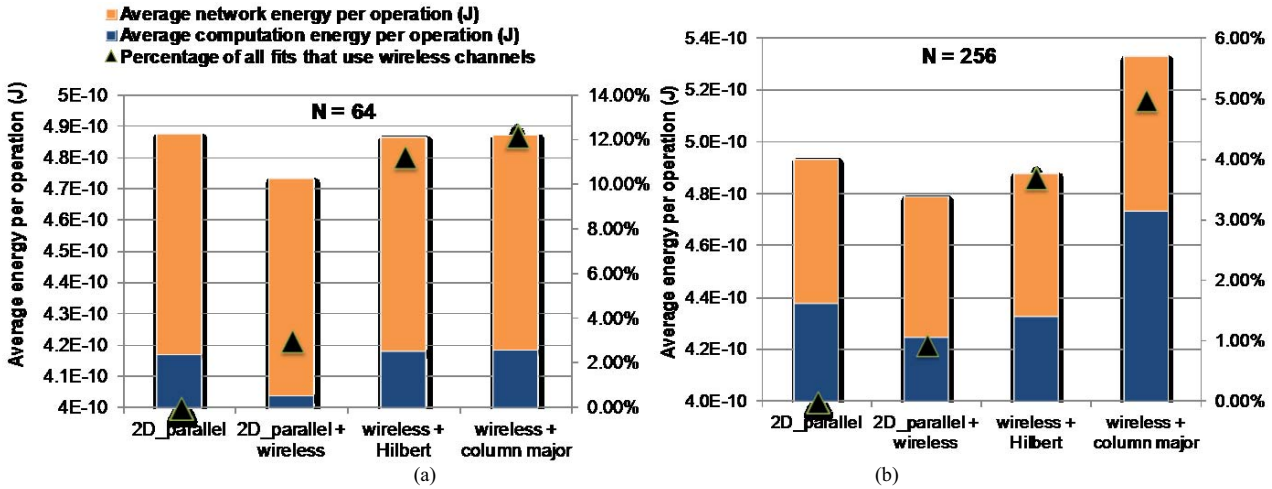

Figure 6. Proportion of flits using wireless shortcuts and energy consumption across network architectures and system sizes

wireless shortcuts. Note that the number of shortcuts (three) is much lower than the number of nodes (64, 256) in the system. As expected, *2D_parallel + wireless*, being a wireless-agnostic allocation method, leads to the lowest proportion of flits using wireless shortcuts. On the other hand, *wireless + column-major* allocation leads to the highest proportion of flits using wireless shortcuts across system sizes. This is because it is a wireless-aware allocation method, in which the partitions that do not get direct access to wireless shortcuts are aligned with the shortcuts, providing them with access to the shortcuts during routing, as explained earlier in Section III-D-3.

E. Energy Consumption

In order to determine which architecture is the most energy-efficient, we evaluated the energy spent per operation. This consists of the computation energy component spent within the computation nodes, and the network energy component spent in the network switches, wireless transceivers and wired links. Fig. 6 shows a comparison of the energy spent per operation across different network architectures and system sizes. *2D_parallel + wireless* is the most energy-efficient in terms of overall energy consumption per operation. A closer look reveals that for N=64, the network energy component is indeed lower for the wireless-aware methods, *wireless + Hilbert* and *wireless + column-major*, due to a larger proportion of their flits using wireless shortcuts, each of which consumes less energy than a wired link. However, as mentioned earlier, these suffer from larger communication latencies, thereby making computation nodes wait for longer. Since the contribution of the computation energy component is greater, the overall energy per operation turns out to be higher. The network energy advantage is lost as system size increases (e.g. N=256) because the proportion of flits using wireless shortcuts becomes low across all architectures, and the saving in energy due to flits using wireless shortcuts is more than offset by the additional energy consumption in the wired links.

V. Conclusion

This paper demonstrates a novel use of on-chip networks and on-chip wireless shortcuts in multicore systems to achieve computation throughput of over 10^{11} log/exp operations per second for a class of scientific applications involving concurrently-executing jobs of similar nature but variable computational footprint, while consuming ~0.5 nJ for each such operation. We demonstrate our performance on a widely-used biocomputing application, and explore different methods of allocating these jobs to the computational nodes to achieve high computation throughput in a single multicore chip.

Acknowledgment

This work was supported by NSF grant IIS-0916463.

References

[1] Amazon Elastic Compute Cloud (aws.amazon.com/ec2/) Last accessed 12 Dec. 2012.

[2] S. V. Tota et al, "A Case Study for NoC-Based Homogeneous MPSoC Architectures," *IEEE Trans. VLSI Sys.*, vol.17, no.3, pp.384-388, March 2009

[3] U. Y. Ogras and R. Marculescu, ""It's a Small World After All": NoC Performance Optimization Via Long-Range Link Insertion," *IEEE Trans. VLSI Sys.*, vol. 14, no. 7, pp. 693-706, July 2006.

[4] A. Ganguly et al., “Scalable Hybrid Wireless Network-on-Chip Architectures for Multi-Core Systems,” *IEEE Trans. Comput.*, vol. 60, no. 10, pp. 1485-1502, Oct. 2011.

[5] The CIPRES Science Gateway (www.phylo.org/sub_sections/portal/) Last accessed 12 Dec. 2012.

[6] National Center for Biotenchonology Information (www.ncbi.nlm.nih.gov/guide/all/#tools_) Last accessed 12 Dec. 2012.

[7] Earth System Modeling Framework (www.earthsystemmodeling.org/) Last accessed 12 Dec. 2012.

[8] A. Kalyanaraman, “Algorithms for genome assembly” in *Encyclopedia of Parallel Computing*, ed. D. Padua , Springer Science+Business Media LLC, DOI 10.1007/978-0-387-09766-4, In Press, 2011.

[9] T. Majumder, P. Pande and A. Kalyanaraman, “Accelerating Maximum Likelihood Based Phylogenetic Kernels Using Network-on-Chip,” in *Proc. 23rd Intl. Sym. Comp. Arch. High Perf. Comp. (SBAC-PAD)*, 26-29 Oct. 2011, pp. 17-24.

[10] F. Trivino et al., “Exploring NoC Virtualization Alternatives in CMPs,” in *Proc. 20th Euromicro Intl. Conf. Comp. & Process. (HW/SW)*, 2012, pp. 473-482.

[11] A. Bakhoda, J. Kim and T. M. Aamodt, “Throughput-effective on-chip networks for manycore accelerators,” in *Proc. 43rd Ann. IEEE/ACM Intl. Sym. MicroArch. (MICRO)*, 2010, pp. 421-432.

[12] C.-L. Chou and R. Marculescu, “Contention-aware application mapping for network-on-chip communication architectures,” in *Proc. IEEE Intl. Conf. Comput. Des. (ICCD)*, 2008, pp. 164-169.

[13] B.-G. Nam, H. Kim, and H.-J. Yoo, “Power and area-efficient unified computation of vector and elementary functions for handheld 3-D graphics systems,” *IEEE Trans. Comput.*, vol. 57, no. 4, pp. 490–504, Apr. 2008.

[14] Circuits Multi-Projects, 46, Avenue Félix Viallet, 38031 GRENOBLE FRANCE (cmp.imag.fr) Last accessed 12 Dec. 2012.

[15] R. Marculescu, U. Y. Ogras, L.-S. Peh, N. E. Jerger, and Y. Hoskote, “Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 1, pp. 3–21, Jan. 2009.

[16] K. Kempa, et al., “Carbon Nanotubes as Optical Antennae,” *Advanced Materials*, vol. 19, 2007, pp. 421-426.

[17] D. Hilbert, "Über die stetige Abbildung einer Linie auf ein Flächenstück", *Mathematische Annalen*, vol. 38,no. 3, pp. 459-460, 1891.

[18] P. P. Pande, et al., “Performance evaluation and design trade-offs for network-on-chip interconnect architectures,” *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[19] W. J. Dally, C. L. Seitz, “Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,” *IEEE Trans. Comput.*, vol.C-36, no.5, pp.547-553, May 1987.

[20] A. Stamatakis, “RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models,” *Bioinformatics*, vol. 22, no. 21, pp. 2688-2690, Nov. 2006.

[21] The Exelixis Lab, Heidelberg Institute for Theoretical Studies, Heidelberg, Germany (sco.h-its.org/exelixis/software.html) Last accessed 12 Dec. 2012.