Wireless NoC Platforms With Dynamic Task Allocation for Maximum Likelihood Phylogeny Reconstruction

Turbo Majumder
Indian Institute of Technology Delhi and
Washington State University

Partha Pratim Pande and Ananth Kalyanaraman
Washington State University

Editor's notes:
Maximum likelihood (ML) phylogeny is an important statistical approach in computational biology that estimates the most likely evolutionary relationship among a given set of species. This paper demonstrates how wireless network-on-chip (WiNoC)-based multicore platforms can be employed to achieve faster time-to-solution ML phylogeny reconstruction.

—Paul K. Ampadu, MIT

Phylogenetic inference is one of the grand challenge problems in bioinformatics. Its aim is to reconstruct an evolutionary tree given a set of n taxa (species). In the reconstructed tree, the taxa form the leaves, and the branches indicate divergence from a common ancestor. The key to reconstruction is observing and characterizing variations at the DNA and protein level. Phylogeny reconstruction can be broadly categorized into the following: distance-based hierarchical methods [e.g., neighbor joining (NJ)], combinatorial optimization using maximum parsimony (MP), and statistical estimation methods [e.g., maximum likelihood (ML), Bayesian inference (BI)]. Of these, the estimation approaches such as ML and BI are statistically consistent and are therefore widely used [1]. These methods provide a statistical likelihood score for each reconstructed tree using the phylogenetic likelihood function (PLF) [2]. The boost in quality, however, comes at a high computation cost as the ML formulation is NP-hard [3] and suffers from the need to explore a super-exponential (in n) number of trees. For example, a run using RAxML [4], which is one of the most widely used programs to compute ML-based phylogeny, on an input comprising 1500 genes has been reported to take up to 2.25 million CPU hours on the IBM BlueGene/L supercomputer [5], which has been parallelized at a coarse level (using compute clusters) to finish in 14 h using 1024 CPUs. With increasing availability of genomic data, as documented in public genomic data banks such as the National Center for Biotechnology Information (http://www.ncbi.nlm.nih.gov/guide/dna-rna/), the relevance and the utility of the statistical estimation approaches are only expected to grow. To realize their full potential, scalable methods that use novel combinations of algorithmic heuristics, hardware acceleration, and high-performance computing are needed. An effective way to harness the compute power within the hardware acceleration units of modern day multicore architectures is to couple them with software-level parallelization—i.e., run the software on a conventional CPU and offload the computation-heavy tasks to the accelerator through an interface like PCIe.

Digital Object Identifier 10.1109/MDAT.2013.2288778
Date of publication: 05 November 2013; date of current version: 22 July 2014.

The advantage of using conventional NoC-based platforms to accelerate ML applications has been shown in [6], where we evaluated the performances of conventional wireline 2-D and 3-D NoCs. Introduction of long-range links in regular 2-D architectures like mesh have been shown to reduce the overall network diameter and improve intercore communication latency [7]. Use of on-chip wireless links to implement these shortcuts has been shown to generate significant savings in latency and energy in presence of standard benchmarks, even considering the overhead of wireless transceivers [8]. In this paper, we design and evaluate wireless-NoC (WiNoC)-enabled many-core platforms to accelerate ML phylogeny reconstruction applications by targeting computationally intensive function kernels. We explore different approaches of allocating these jobs to the accelerator, determine the overheads, and evaluate the tradeoffs. The targeted kernels are sped up by over 2000× (application speedup, considering all relevant overheads) for a fairly large input, and the total runtime (including the nonaccelerated portion running in software and data transfer overheads) gets reduced by more than 80%. The platforms are also energy efficient, consuming ~0.5 nJ per arithmetic operation. We also demonstrate how our NoC architectures would scale with a higher computational footprint per kernel.

Related work

Considerable work has been done on designing hardware accelerators for the different approaches of phylogeny reconstruction. These efforts have generally targeted MP, ML, and BI because of their wide usage and large time complexities.

A detailed survey of various hardware acceleration approaches for biocomputing is provided in [9]. A parallelized version of breakpoint-median phylogeny using both software and FPGA was shown to achieve a speedup of 417× for a whole-genome phylogeny reconstruction. In [10], we implemented a NoC-based accelerator that achieves a speedup of 774×.

For BI, hardware acceleration has been proposed using cell broadband engine (CBE), GPU, and general purpose multicores and FPGA. These platforms achieve an order of magnitude speedup over software [9].

For ML phylogeny, which is the target application in this paper, a genetic algorithm using a hybrid hardware–software approach achieves an overall speedup of 30×. A CBE-based implementation for RAxML is shown to outperform the software by 2×. Another implementation of RAxML using FPGA boards with built-in DSP slices achieves a speedup of 8×. We presented 2-D and 3-D wireline NoC-based hardware accelerators [6] that achieved application speedups (considering all relevant overheads) over 900× and 1000×, respectively, and overall runtime reduction more than 5×, while being energy efficient.

NoCs have been shown to perform better by insertion of long-range wired links following principles of small-world graphs [7]. Although there are significant performance gains, the use of wired links beyond a certain length has been shown to be less energy efficient than the use of on-chip wireless links. Several standard traffic patterns have been explored using WiNoC architectures in [8]. In this work, we leverage the benefits of using long-range wireless shortcuts on a 2-D NoC to design energy-efficient hardware accelerators for ML phylogeny reconstruction, delivering an application speedup over 2000× and energy efficiency of ~0.5 nJ per arithmetic operation.

In our design, we implement logarithmic calculations in hardware. Fast calculation of logarithms in hardware has been a well-researched topic. A brief discussion on existing work in this area can be found in [6]. Here, we build upon the unified computation architecture for calculating elementary functions presented in [11], which uses a fixed-point hybrid number system (FXP–HNS) to integrate all operations in a power- and area-efficient manner with a low percentage of error.

Design of NoC with long-range links

We present the design of a multicore SoC, 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.

Processing element (PE)

ML applications typically involve millions of small task kernels that carry out node-level likelihood computations. These computations involve vector products, and logarithm/antilogarithm computations (to obtain log likelihood values). We designed a six-stage, pipelined computation core to carry out these computations efficiently. Details of the computation core design can be found in [6]. We used FXP–HNS [11], 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.

Each PE consists of four computation cores, because the input vector sizes in each task kernel are multiples of four. In our implementation, for the demonstration case of RAxML [4], in the three task kernels collectively accounting for more than 85% of the software runtime, these vector sizes are 8, 12, and 24. As such, their computation requires 2, 3, and 6 PEs, respectively. We call these functions $f2$, $f3$, and $f6$, respectively, to indicate their computational footprint [6]. In addition, each PE has 2 MB of memory in the form of register banks to store inputs and computation outputs. The number of PEs represents the system size N of the multicore system. 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 (http://cmp.imag.fr). The critical path in the computation core determines the clock frequency of 1 GHz.

Wired network architecture

ML applications spawn a stream of independent jobs (function kernels) that individually require variable amounts of computation resources. Communication occurs only among computation nodes (PEs) 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. We implement a wired folded torus where all internode links 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 in the PE computation core. Since our datapath is 64 bit wide, we split each internode message into three 64-bit flits—header, body, and tail. As a result, each internode link needs a minimum bandwidth of 64 Gb/s.

Long-range on-chip wireless links

Our aim is to minimize the average distances among nodes catering to one job. We try to achieve this goal through: 1) use of on-chip wireless shortcuts; and 2) intelligent dynamic node allocation methods. The latter is described in the Dynamic Node Allocation section. As will be seen there, owing to the nature of the application, 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.

Physical layer. Suitable on-chip antennas are necessary to establish the wireless links. It has been shown that for some standard traffic patterns, wireless NoCs designed using carbon nanotube (CNT) antennas can outperform conventional wired counterparts significantly [8]. Antenna characteristics of CNTs in the terahertz frequency range have been investigated both theoretically and experimentally [12]. 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 is possible by using CNTs of different lengths, which are multiples of the wavelengths corresponding to the respective carrier frequencies. Using current technology, it is possible to create 24 nonoverlapping wireless channels, each capable of sustaining a data rate of 10 Gb/s using CNT antennas, details of which are discussed in [8]. The number of wireless links in our system is determined by the bandwidth each link needs to support. As mentioned earlier, each wireless (internode) link needs to sustain a bandwidth of 64 Gb/s. Based on the capacity of the wireless channels (10 Gb/s), we need seven channels per link (providing up to 70-Gb/s bandwidth).

Consequently, the maximum number of single-hop wireless links that we can allow with the current technology is int$(24/7) = 3$.

Link placement. The traffic pattern generated by an application determines the most appropriate locations for placement of wireless links. However, such a traffic-reliant approach is not suitable here, because the sets of communicating PEs (pertaining to the execution of a single kernel) change their location on the network with time. The overhead of assigning wireless links for every change in the application map ($\sim 10^{-7}$ s) is very high. As it is not possible to predict a standard traffic pattern, we observed and analyzed long-term traffic statistics. Observation of traffic patterns across numerous application maps has shown that among nodes that are not colocated (internode 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 the Dynamic Node Allocation section divides the network into four quadrants and the need for long-range links arises when allocated nodes are noncontiguous and lie in neighboring quadrants, the mean distance between which is equal to the diameter, as shown in Figure 1.

Note that we are constrained by only three wireless links due to current technology limitations, as explained earlier. 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 Figure 1.

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*. When a job requests computation resources, *MasterController* allocates the requisite number of

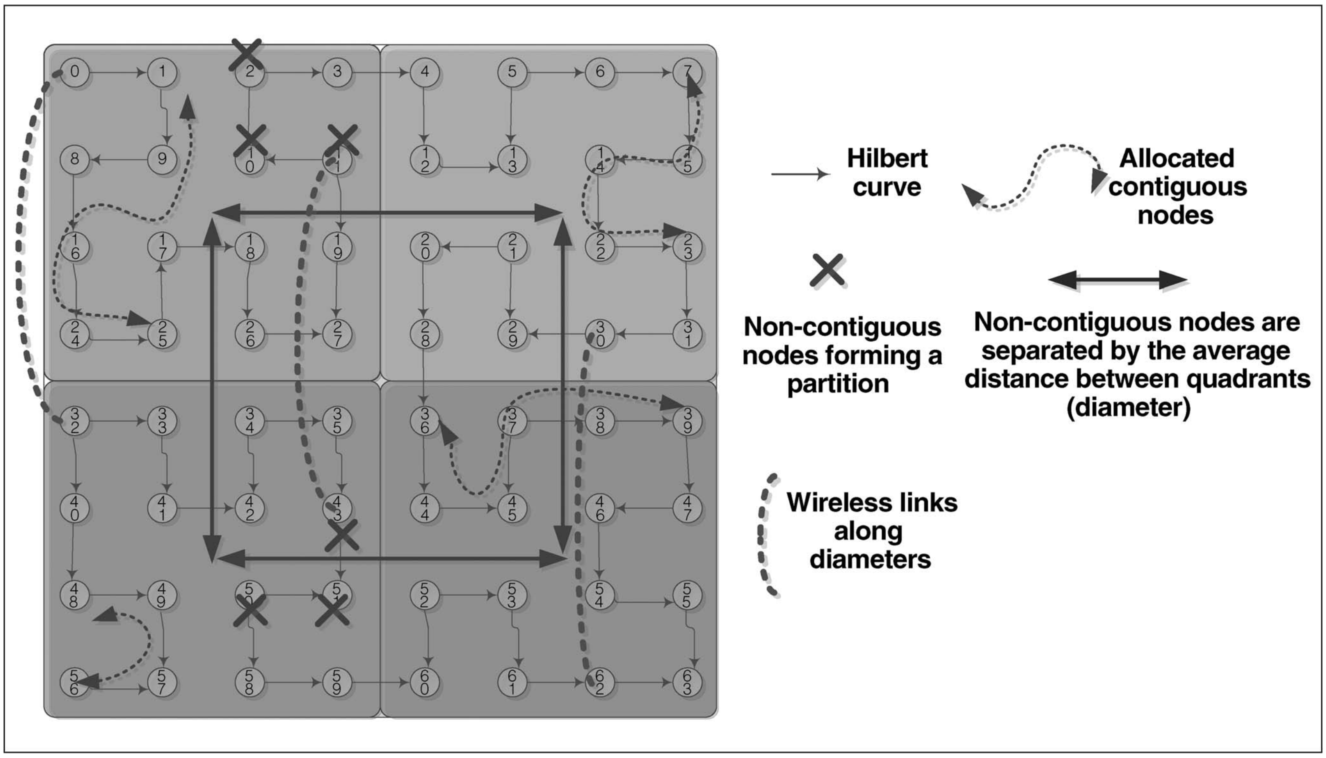
Hilbert curve
Allocated contiguous nodes
Non-contiguous nodes forming a partition
Non-contiguous nodes are separated by the average distance between quadrants (diameter)
Wireless links along diameters

Figure 1. Noncontiguous nodes and long-range communication requirements leading to wireless link placement along diameters.

available computation nodes from the system. The nodes thus allocated form a partition during the course of function execution and communicate with one another. As mentioned earlier, we need intelligent dynamic node allocation methods to ensure colocality of the nodes in a partition. We also have to make sure that these methods do not incur a large allocation time overhead. Simple approaches like breadth-first search do not fit these criteria. We developed the following allocation methods, which can be classified into wireless-agnostic and wireless-aware methods. We also make use of the locality-preserving, space-filling Hilbert curve (see Figure 1) for allocation. The resultant allocated partitions 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.

Parallel best-fit allocation using multiple Hilbert curves. This allocation strategy preferentially looks for a partition with contiguous nodes to maximize colocality, 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 rightangle 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 (as shown in Figure 1). *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.

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."

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 Figure 1, 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."

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 colocality among the allocated nodes. The only advantage of this allocation method is the simplicity of *MasterController* logic and fewer cycles spent in allocation.

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 [13]. 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 four virtual channels. We use deadlock-free e-cube routing in torus [14].

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 and is available to the 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 [7] when involving wireless shortcuts, and e-cube routing when following wired-only paths between a source and a destination.

Experimental results

Experimental setup

The computation core has a datapath width of 64 bits and provides a number representation accuracy of $\sim 10^{-15}$. As mentioned earlier, each PE in the system consists of four computation cores. We synthesized Verilog RTLs for the PEs, the network switches and *MasterController* with 65-nm standard cell libraries from CMP. We used a clock period of 1 ns constrained by the critical path occurring in the core datapath as mentioned in the Processing Element (PE) section. 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 Gb/s each, providing a total link bandwidth of 70 Gb/s. In this work, we considered CNT-antenna-based wireless link design using the technology described in [8]. We considered 0.33 pJ/bit energy dissipation as reported in that work as the energy consumed by each wireless link.

We implemented each of the dynamic node allocation methods mentioned in the Dynamic Node Allocation section. We used a system size of $N = 256$ in our experiments. We model the NoC-based multicore platform as a coprocessor 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 Gb/s. We use a 32-lane PCIe 2.0 for our simulation.

We selected an ML-based phylogenetic reconstruction software called RAxML version 7.0.4 (http://sco.h-its.org/exelixis/software.html) 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 $f6$ (*newviewGTRGAMMA*), $f3$ (*coreGTRCAT*), and $f2$ (*newviewGTRCAT*), respectively, based on the computation resources (the number of computation nodes or PEs) they need for execution. We ran RAxML on some inputs that are provided with the suite. These inputs comprised DNA sequences originally derived from a 2177-taxon 68-gene mammalian data set described in [15]. An input x_y is a set of x aligned sequences (taxa) each y characters long. For example, the input 50_5000 consists of 50 sequences with 5000 characters in each.

Performance

We demonstrate the performance of our WiNoC-based hardware accelerator in Figure 2 and Table 1 by providing a detailed breakdown of different times while running RAxML. The time spent in running RAxML using four threads on a Pentium IV-based server is used as the baseline. When using the hardware accelerator, a portion of this runtime has serial dependency or contains functions not targeted by the accelerator (t_x) and the other portion is taken care of by the accelerator (t_u). The time spent by the accelerator in actually computing those function kernels is referred to as t_h, the time spent in allocating these function kernels is referred to as t_a, and the time spent in data transfer over PCIe is referred to as t_p. Clearly, application speedup is

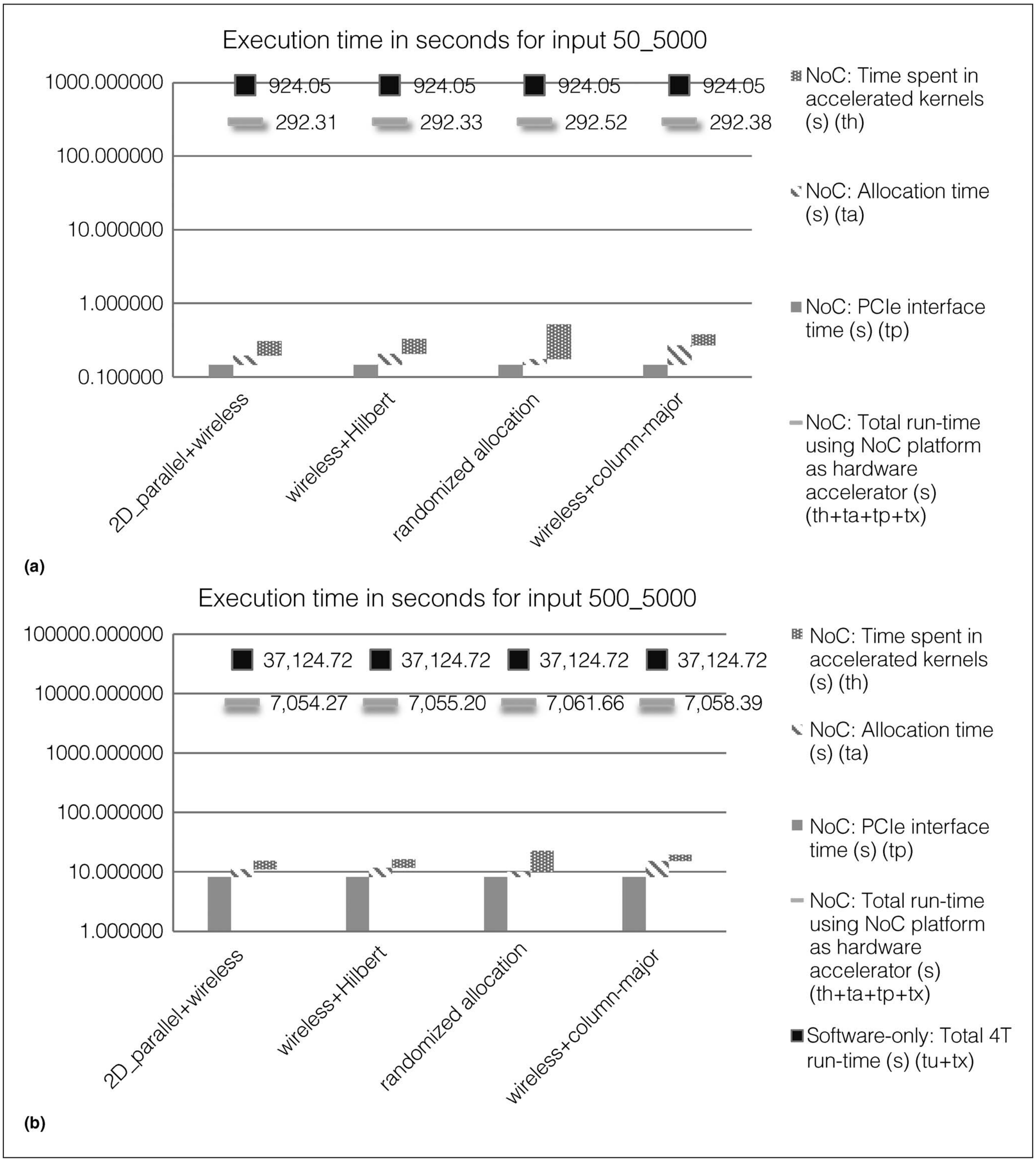

Figure 2. Breakdown of different times while running RAxML with two different inputs on our WiNoC-based hardware accelerator. Definitions of th, ta, tp, tu and tx are provided in the Performance section.

given by $t_u/(t_h + t_a + t_p)$, and the overall runtime reduction is given by $(t_u + t_x)/(t_h + t_a + t_p + t_x)$. Referring to Figure 2, we note that we obtain over 2000× application speedup for input 50_5000 and 2D_parallel+wireless allocation. For this case, from Table 1, we have $t_u = 632.052$ s, $t_h = 0.113$ s,

Table 1 Breakup of the different components of time spent while using a wireless NoC-based accelerator *vis-a-vis* only software.

	NoC Architecture	NoC: Time spent in accelerated kernels (s) (th)	NoC: Allocation time (s) (ta)	NoC: PCIe interface time (s) (tp)	NoC: Software run-time of portion not offloaded to NoC (s) (tx)	**NoC: Total run-time using NoC platform as hardware accelerator (s) (th+ta+tp+tx)**	Software-only: Total 4T run-time (s) (tu+tx)	Software only: Run-time of tasks offloaded to NoC (s) (tu)	**Application Speedup (tu/(th+ta+tp))**
Input: 50_5000	**2D_parallel + wireless**	0.113	0.050	0.145	292.000	**292.309**	924.052	632.052	**2050.132**
	wireless + Hilbert	0.123	0.060	0.145	292.000	**292.328**	924.052	632.052	**1927.566**
	randomized allocation	0.341	0.030	0.145	292.000	**292.516**	924.052	632.052	**1224.976**
	wireless + column-major	0.112	0.123	0.145	292.000	**292.380**	924.052	632.052	**1663.746**
Input: 500_5000	**2D_parallel + wireless**	4.293	2.859	8.273	7038.848	**7054.273**	37124.723	30085.876	**1950.380**
	wireless + Hilbert	4.640	3.436	8.273	7038.848	**7055.198**	37124.723	30085.876	**1840.106**
	randomized allocation	12.807	1.728	8.273	7038.848	**7061.656**	37124.723	30085.876	**1319.057**
	wireless + column-major	4.258	7.007	8.273	7038.848	**7058.386**	37124.723	30085.876	**1539.819**

$t_a = 0.05$ s, and $t_p = 0.145$ s, which gives us an application speedup of 2050×. The same metric in [6] evaluates to a best-case application speedup of 1061× for a 3-D torus NoC, and 908× for a 2-D wired torus NoC. Comparing the different allocation methods in Figure 2 and Table 1, we observe that randomized allocation indeed has the lowest allocation time (t_a) as expected, but there is a much lower degree of colocality in the allocated partitions, leading to poorer acceleration. The total time spent in node allocation for the methods discussed in this paper (e.g., $t_a = 2.859$ s for 2D_parallel+wireless as shown in Table 1) compare favorably with the methods proposed for 3-D NoC in [6] (e.g., $t_a = 3.305$ s for 3-D torus NoC). On the other extreme, the time spent in accelerated kernels (t_h) is the lowest in wireless+column-major, but a lot of time overhead is spent in allocation because we cannot leverage the advantages of the Hilbert curve. As such, the application speedup is worse than both wireless+Hilbert and 2D_parallel+wireless. Note that the time spent in data transfer via PCIe is independent of the accelerator architecture and is a function of input size only.

In terms of energy efficiency, 2D_parallel+wireless is the most efficient, as explained in detail in [16], with each method consuming ~0.5 nJ per operation. For the test cases 50_5000 and 500_5000, across different allocation methods, this translates to a total energy consumption in the range of 25–27.8 J and 949–1060 J, respectively. The energy spent by *MasterController* is between 0.15 mJ (for randomized allocation) and 0.49 mJ (for wireless + column-major) for 50_5000, and between 8.7 mJ (for randomized allocation) and 28 mJ (for wireless_column-major) for 500_5000. Its area footprint is ~0.024 mm^2. This makes its energy and area overhead negligible with respect to that of the system.

Scaling computation footprint

In order to investigate how our WiNoC-based platforms respond to different computational footprints, we note that our application model subsumes several task kernels being simultaneously executed. Keeping the system size constant at $N = 256$, we can increase the footprint of each kernel while still allowing for a large number of kernels to execute simultaneously. We choose three kernels, now with

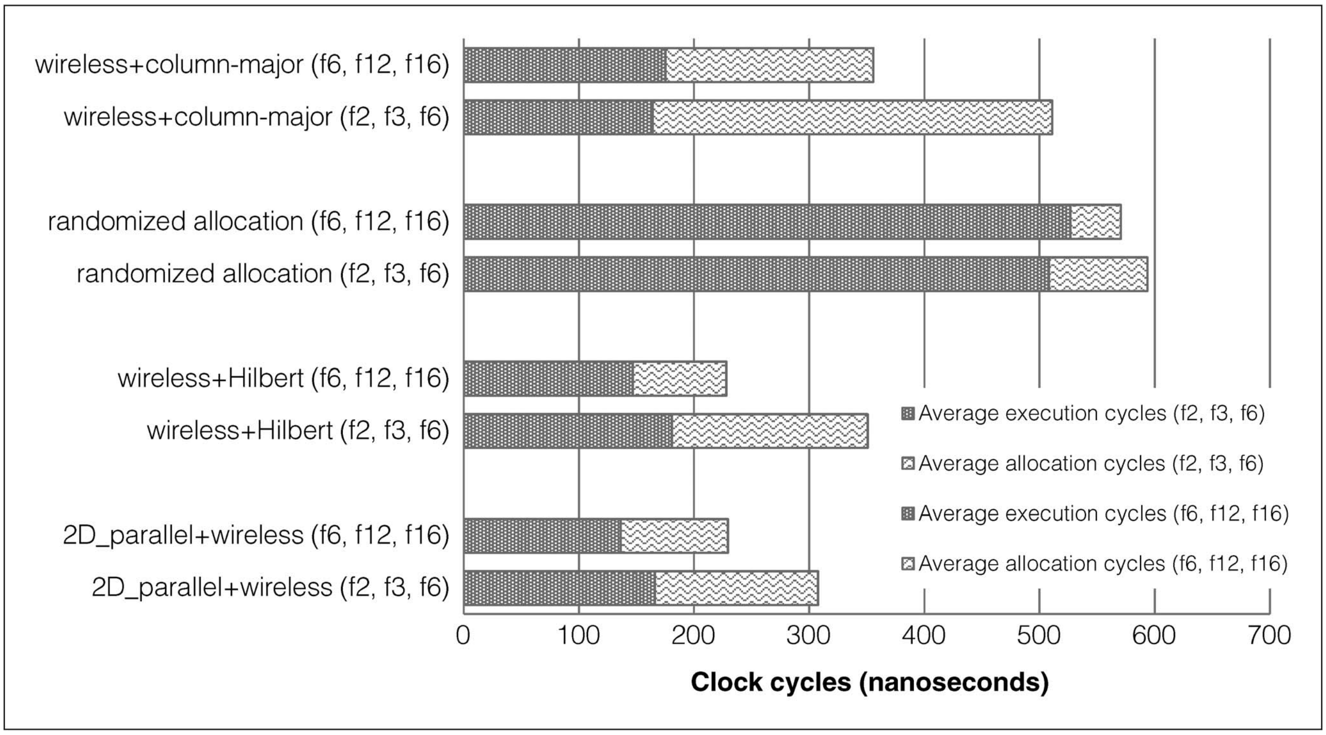

Figure 3. Comparison of allocation and execution time across dynamic allocation methods. $(\boldsymbol{fx}, \boldsymbol{fy}, \boldsymbol{fz})$ **denotes a combination of tasks with computational footprints of *x*, *y*, and *z* nodes, respectively.**

(larger) footprints of 6 (*f*6), 12 (*f*12), and 16 (*f*16) PEs, build a large number of application maps with these, and compare the performance with the RAxML kernels (*f*2, *f*3, and *f*6). The comparison with respect to the number of clock cycles spent in allocation of partitions and execution time (time to completion) is shown in Figure 3. Note that the allocation time reduces in all the methods because with larger partition sizes, we have fewer partitions to allocate. The reduction is most prominent in wireless+column-major and wireless+Hilbert methods. However, with larger partitions, we observe the execution time to go up for wireless+column-major and randomized allocation. This is because these two methods have little or no focus on ensuring colocality while task allocation and larger partitions end up having more dispersed nodes. In general, we note that with computational footprints scaling up, both methods using Hilbert curves tend to perform at par.

Our paper demonstrates the design of a NoC-based many-core chip that accelerates targeted functions in a computation-intensive bioinformatics application, viz., ML phylogeny reconstruction, and can be easily extended to similar biocomputing applications. Our NoC design incorporates on-chip wireless shortcuts, and we propose and evaluate various schemes to allocate tasks on the many-core platform. Our experiments show that the use of space-filling Hilbert curve provides greater colocality of dynamically allocated computation nodes, particularly when the computational footprint scales up. For the ML application we used as a demonstration case, we achieved over 2000× application speedup. ■

Acknowledgment

This work was supported by the National Science Foundation (NSF) under Grant IIS-0916463.

■ References

[1] C. R. Linder and T. Warnow, An overview of phylogeny reconstruction,'' in *Handbook of Computational Molecular Biology*, S. Aluru, Ed. London, U.K.: Chapman & Hall/CRC Press, 2005, ch. 19, ser. Computer and Information Science.

[2] J. Felsenstein, ''Evolutionary trees from DNA sequences: A maximum likelihood approach,''

J. Molecular Evol., vol. 17, pp. 368–376, 1981.

[3] B. Chor and T. Tuller, ''Maximum likelihood of evolutionary trees: Hardness and approximation,'' *Bioinformatics*, vol. 21, no. 1, pp. 97–106, 2005.

[4] A. Stamatakis, ''RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models,'' *Bioinformatics*, vol. 22, pp. 2688–2690, 2006.

[5] N. Alachiotis, E. Sotiriades, A. Dollas, and A. Stamatakis, ''Exploring FPGAs for accelerating the phylogenetic likelihood function,'' in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2009, DOI: 10.1109/IPDPS.2009.5160929.

[6] T. Majumder, M. Borgens, P. Pande, and A. Kalyanaraman, ''On-chip network-enabled multi-core platforms targeting maximum likelihood phylogeny reconstruction,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* vol. 31, no. 7, pp. 1061–1073, Jul. 2012.

[7] U. Y. Ogras and R. Marculescu, ''It's a small world after all'': NoC performance optimization via long-range link insertion,'' *IEEE Trans. Very large Scale Integr. (VLSI) Syst.*, vol. 14, no. 7, pp. 693–706, Jul. 2006.

[8] 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.

[9] S. Sarkar, T. Majumder, A. Kalyanaraman, and P. Pande, ''Hardware accelerators for biocomputing: A survey,'' in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 3789–3792.

[10] T. Majumder, S. Sarkar, P. Pande, and A. Kalyanaraman, ''NoC-based hardware accelerator for breakpoint phylogeny,'' *IEEE Trans. Comput.*, vol. 61, no. 6, pp. 857–869, Jun. 2012.

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

[12] K. Kempa *et al.*, ''Carbon nanotubes as optical antennae,'' *Adv. Mater.*, vol. 19, pp. 421–426, 2007.

[13] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, ''Performance evaluation and design trade-offs for network-on-chip interconnect architectures,'' *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 1025–1040, Aug. 2005.

[14] W. J. Dally and C. L. Seitz, ''Deadlock-free message routing in multiprocessor interconnection networks,'' *IEEE Trans. Comput.*, vol. C-36, no. 5, pp. 547–553, May 1987.

[15] O. R. P. Bininda-Emonds *et al.*, ''The delayed rise of present-day mammals,'' *Nature*, vol. 446, pp. 507–512, 2007.

[16] T. Majumder, P. P. Pande, and A. Kalyanaraman, ''High-throughput, energy-efficient network-on-chip-based hardware accelerators,'' *Sustain. Comput., Inf. Syst.*, vol. 3, no. 1, pp. 36–46, Mar. 2013.

Turbo Majumder is an Assistant Professor in the Department of Electrical Engineering, Indian Institute of Technology Delhi, New Delhi, India. He works on many-core network-on-chip platforms, system-on-chip platforms, hardware acceleration, and high-performance computing. Majumder has a BS in electronics and electrical communication engineering, an MS in automation and computer vision from Indian Institute of Technology Kharagpur, West Bengal, India, and a PhD in electrical engineering from Washington State University, Pullman, WA, USA. He is a member of the IEEE.

Partha Pratim Pande is an Associate Professor and the holder of the Boeing Centennial chair in Computer Engineering at the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA. His current research interests are novel interconnect architectures for multicore chips, on-chip wireless communication networks, and hardware accelerators for biocomputing. Pande has an MS in computer science from the National University of Singapore, Singapore and a PhD in electrical and computer engineering from the University of British Columbia, Vancouver, BC, Canada. He is a Senior Member of the IEEE.

Ananth Kalyanaraman is an Associate Professor at the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA. His research interest is in high-performance computational biology. Kalyanaraman has a PhD in computer engineering from Iowa State

University, Ames, IA, USA (2006). He is a member of the American Association for the Advancement of Science (AAAS), the Association for Computing Machinery (ACM), the IEEE Computer Society (IEEE/CS), and the International Society for Computational Biology (ISCB).

■ Direct questions and comments about this article to Turbo Majumder, Department of Electrical Engineering, Indian Institute of Technology Delhi, New Delhi 110016, India, phone +91 11 2659 1073; fax +91 11 2658 1606; turbo@ee.iitd.ac.in; http://web.iitd.ac.in/~turbo/.