Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2015, Article ID 902591, 20 pages
http://dx.doi.org/10.1155/2015/902591

Research Article

Performance Analysis of Homogeneous On-Chip Large-Scale Parallel Computing Architectures for Data-Parallel Applications

Xiaowen Chen,[1,2] Zhonghai Lu,[2] Axel Jantsch,[3] Shuming Chen,[1] Yang Guo,[1] Shenggang Chen,[1] and Hu Chen[1]

[1]*College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China*
[2]*Department of Electronic Systems, KTH-Royal Institute of Technology, Kista, 16440 Stockholm, Sweden*
[3]*Institute of Computer Technology, Vienna University of Technology, 1040 Vienna, Austria*

Correspondence should be addressed to Xiaowen Chen; xiaowenc@kth.se

Received 28 August 2014; Revised 18 January 2015; Accepted 18 January 2015

Academic Editor: Dimitrios Soudris

On-chip computing platforms are evolving from single-core bus-based systems to many-core network-based systems, which are referred to as *On-chip Large-scale Parallel Computing Architectures (OLPCs)* in the paper. Homogenous OLPCs feature strong regularity and scalability due to its identical cores and routers. Data-parallel applications have their parallel data subsets that are handled individually by the same program running in different cores. Therefore, data-parallel applications are able to obtain good speedup in homogenous OLPCs. The paper addresses modeling the speedup performance of homogeneous OLPCs for data-parallel applications. When establishing the speedup performance model, the network communication latency and the ways of storing data of data-parallel applications are modeled and analyzed in detail. Two abstract concepts (*equivalent serial packet and equivalent serial communication*) are proposed to construct the network communication latency model. The uniform and hotspot traffic models are adopted to reflect the ways of storing data. Some useful suggestions are presented during the performance model's analysis. Finally, three data-parallel applications are performed on our cycle-accurate homogenous OLPC experimental platform to validate the analytic results and demonstrate that our study provides a feasible way to estimate and evaluate the performance of data-parallel applications onto homogenous OLPCs.

1. Introduction and Motivation

As technology advances, on-chip computing platforms are evolving from single-core bus-based systems to many-core network-based systems, which feature integrating a number of computing cores that run in parallel and adopting an on-chip network that provides concurrent pipelined communication. The many-core network-based systems are referred to as *On-chip Large-scale Parallel Computing Architectures (OLPCs)* in the paper. OLPCs can be highly homogeneous or irregular and heterogeneous. Homogenous OLPC owns its characteristics of strong regularity and scalability, since processor cores and routers in it are the same. Each processor core has the same computation capability. As one way of parallel processing, data parallelism partitions data into several blocks that are mapped to different processors and processors work in SPMD (Single Program Multiple Data) mode, that is, they handle their own data blocks by running the same program. Data-parallel applications have the parallel data set that can be partitioned in parallel into data subsets and each data subset can be handled individually by the same program and has marginal synchronization overhead, so they are well scalable and can be used to exploit the potential of multiple computing cores. Therefore, homogenous OLPCs and data-parallel applications match each other well. Data-parallel applications are able to obtain good speedup on homogenous OLPCs. Therefore, the focus of the paper is to provide a workable way to estimate and evaluate the performance of homogenous OLPCs with data-parallel applications.

Scalability is one of the important features of homogenous OLPCs. In homogenous OLPCs, as the network size is scaled up, the network communication latency is increasing

and becoming one of the most significant factors affecting the system performance. Therefore, we firstly propose two abstract concepts: *equivalent serial packet* and *equivalent serial communication*, and then we construct a detailed network communication latency model. Then, based on Amdahl's Law, we propose a performance model including the detailed network communication latency. Two traffic models (*uniform* and *hotspot*) are used to reflect the two ways of storing data of data-parallel applications. The uniform traffic model matches the *distributed* way that data are equally distributed into all nodes, while the hotspot traffic model matches the *centralized* way that data are only maintained in the central node. Our models also analyze the performance impact of the noncommunication/communication ratio. Some useful suggestions are presented during the performance model's analysis. Finally, we map three data-parallel applications (Wavefront Computation, Vector Norm, and Block Matching Algorithm in Motion Estimation) on our cycle-accurate homogenous OLPC experimental platform to validate and demonstrate our performance analysis.

The contributions of the paper are summarized as follows.

(1) Since homogenous OLPCs match data-parallel applications well and vice versa, our study exhibits a workable way to formulate and evaluate the speedup performance of data-parallel applications onto homogenous OLPCs before application programming and hardware design.

(2) Two abstract concepts, *equivalent serial packet* and *equivalent serial communication*, are proposed and then used to construct the detailed network communication latency model (see Section 4.3).

(3) Based on Amdahl's Law, we propose a performance model of homogeneous OLPCs for data-parallel applications (see Section 4.4). The proposed performance includes the proposed network communication latency model and adopts two traffic models (*uniform* and *hotspot*) so as to have two forms (see Sections 4.4.1 and 4.4.2). They, respectively, reflect the distributed way and the centralized way of storing data of data-parallel applications.

(4) A cycle-accurate homogenous OLPC experimental platform is built up and three real data-parallel applications are mapped to validate the effectiveness of the proposed performance model.

The rest of the paper is organized as follows. Section 2 presents the background and related work. Section 3 discusses the characteristics of homogenous OLPCs and data-parallel applications and their relationship. Section 4 proposes the communication latency model and the performance model of homogenous OLPCs and details the analysis. Section 5 maps three data-parallel applications on our homogenous OLPC platform to validate the effectiveness of the performance model. Section 6 discusses the applicabilities and the limitations of our performance model. Finally, we conclude in Section 7.

2. Background and Related Work

The development of on-chip computation presents two trends. One is towards a growing number of processors integrated on a chip [1, 2]. it is moving away from a sequential to a parallel paradigm leading to tens, dozens, hundreds, and soon even thousands of computing cores on a single chip. A number of computing cores are potential to cooperate in parallel to obtain higher performance of parallel applications. The other trend is about the interconnection of on-chip resources. The communication infrastructure is developing into a similarly parallel structure, which is often called a Network-on-Chip (NoC) [3–5]. Shared, serial buses are replaced by pipelined communication networks that allow hundreds or thousands of communications going on concurrently at any time. Combining the two trends, on-chip computing platforms are evolving from single-core bus-based systems to many-core network-based systems, which are referred to as *On-chip Large-scale Parallel Computing Architectures (OLPCs)* in the paper. Understanding the speedup potential that OLPC computing platforms can offer is a fundamental question to continually pursuing higher performance.

With respect to performance analysis, Amdahl's Law [6] provides a simple, yet very useful method to evaluate the performance of a parallel system. Its fundamental hypothesis is that the computation problem size does not change when running on enhanced parallel systems. Its main result shows that the percentage of the serial portion dominates the speedup limit. Amdahl's Law is a pessimistic view that the speedup does not increase infinitely along with the increase of the number of paralleled processor cores. Based on Amdahl's Law, many researchers discussed their variants for different purposes. In [7], Li and Malek discussed the effect of noncommunication/communication ratio on the speedup based on Amdahl's Law, but his communication delay model is simple without considering the detail of interconnects. In [8], Paul revisited Amdahl's Law on the single chip heterogeneous multiprocessor. His focus was on the performance impact induced by different types of processor cores with different processing capability. In [9], Cho and Melhem presented the corollaries to Amdahl's Law in order to study the interaction between parallelization and energy consumption. In [10], Hill and Marty offered a corollary of a simple model of multicore hardware resources based on Amdahl's Law. He proved that an enhanced core is necessary for the high system performance but the parallelism supported by systems with such cores suffers. In [11], Loh extended Hill's work to study the performance impact of uncore function units on the multicore system's throughput. In both Hill's and Loh's discussions, the effect of network communication latency is omitted. In OLPCs, the enhancement of application performance may be restricted by the increasing network communication latency, even though the number of cores increases. We note that less work aforementioned discusses the effect of network communication latency on the performance of OLPCs. In the paper, we detail the network communication latency by proposing two abstract concepts, *equivalent serial*

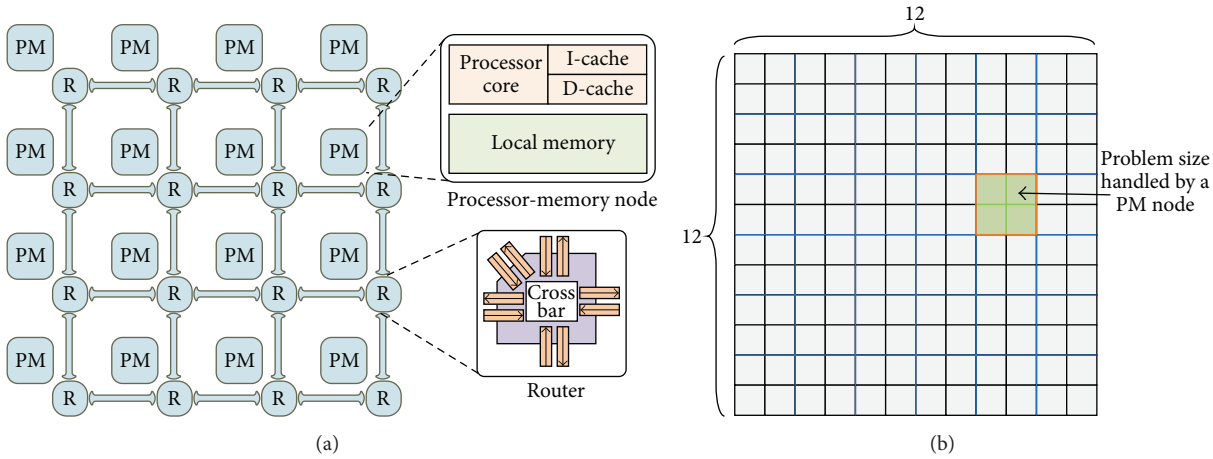

Figure 1: (a) Sketch map of homogenous OLPCs and (b) an example of data partitioning of data-parallel applications.

packet and *equivalent serial communication*, and establish the performance model of homogenous OLPCs. Our model, verified by real data-parallel applications, exhibits a workable way to estimate and evaluate the performance of homogenous OLPCs.

3. Homogenous OLPCs and Data-Parallel Applications

Homogenous OLPCs are a suitable architecture for data-parallel applications and vice versa. Regularity and scalability are the key features of homogenous OLPCs. Figure 1(a) shows an example of homogenous OLPCs. The communication infrastructure is a regular 2D-mesh NoC, which is the most popular NoC topology proposed today [12]. As we can see, the processor type and the local memory volume in each Processor-Memory (PM) node is the same so that each PM node has the same computation capability. All PM nodes are networked by routers. The network size is scalable. As one way of parallel processing, data parallelism partitions data into several blocks that are mapped to different processors. Processors handles their own data blocks by running the same program. Data parallelism is efficient for applications with high computation complexity (e.g., image processing, hydrodynamics computing). These data-parallel applications are well scalable and their data are regular. They are easily parallelized by partitioning their data. Figure 1(b) illustrates a data partitioning way of data-parallel applications. Assuming that there are 144 (12×12) data to be processed by a data-parallel application and the homogenous OLPC is with the network size of 36 (6×6). Since the computation ability of each PM node is the same, it is obvious to partition the 144 data into 36 equivalent parts. Each equivalent part contains 4 sets of data and is handled by a PM node. As the network size is scaled up and hence more PM nodes are included, we can repartition the data to suit the number of PM nodes in order to gain higher performance. However, the network communication limits the performance. We consider two

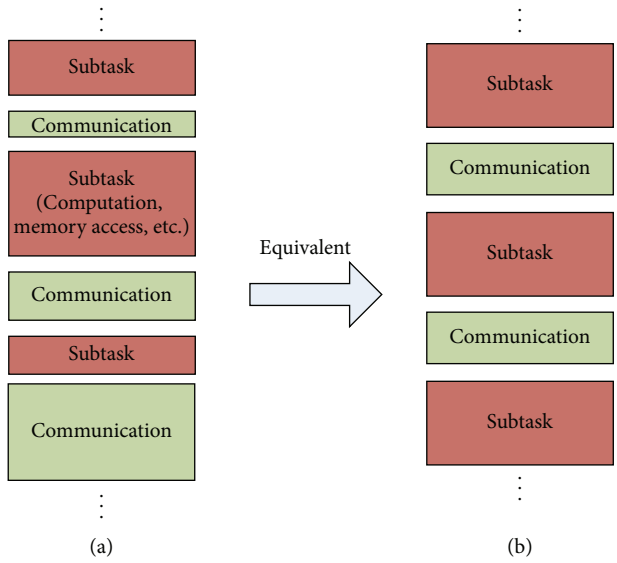

Figure 2: The subprogram running on a processor node is abstracted as a set of subtasks and communications.

traffic models which reflect two ways of storing data of data-parallel applications. The uniform traffic model matches the *distributed* way that data are distributed equally into all local memories of all nodes. The hotspot traffic model matches the *centralized* way that data are only maintained in the central node.

4. Models and Analysis

4.1. Problem Definition. The problem we consider is the performance in the context of homogeneous OLPCs for data parallel applications. We give detailed analysis on communication latency. The program running on OLPCs are divided into several subprograms running on different processor nodes. The subprogram can be abstracted as a set of subtasks and communications (see Figure 2(a)). The communication

Table 1: Calculation of Hop Count in k-ary-2-mesh.

Uniform	Hotspot	
$H = 2\left(\frac{k}{3} - \frac{1}{3k}\right)$	$H = \begin{cases} \frac{k}{2} + \frac{k}{2(k^2-1)k} & k\text{: even} \\ \frac{k}{2} & k\text{: odd} \end{cases}$	

denotes the interaction between two communicating processor nodes. A communication contains one or more packets transmitted in the network. The subtask denotes the noncommunication processing (e.g., computation, memory access, etc.) between two successive communications. To facilitate constructing the models of communication latency and the performance, we make the following three assumptions.

(1) The noncommunication time and communication time of the subprogram assigned to each node is equal to each other. That is, the subprogram in each node contains the same number of subtasks and communications.

(2) The execution time of each subtask is also equal to each other.

(3) The time of each communication is also equal to that of others.

Figure 2(b) is the reabstracted subprogram based on assumption (2) and (3). The sum of the subtasks and communications of Figure 2(b) is equal to that of Figure 2(a).

4.2. Notations. To facilitate the analysis, we first define a set of symbols in Notaions section.

4.3. Communication Latency Model. Communication latency contains two parts: minimal (noncontention) latency and contention latency.

The minimal latency is determined by the distance of the two communicating nodes. We use hop count to calculate the latency. Table 1 lists our calculated hop count following [13]. We consider two representative traffic models (Uniform and Hotspot) in 2D-mesh networks. For hotspot traffic, the central node is chosen as the hotspot node.

The contention latency mainly depends on the behavior of parallel applications running on OLPCs. In general, it is difficult to quantify the contention latency exactly. "When to communicate," "which processor core starts a message passing" and "where the destination is" lead to different contention latency. If no contention occurs, transmitting a packet in one hop takes 1 cycle ($\tau_{1\text{hop}} = 1$) in our experimental platform shown in Figure 8. However, network contention makes $\tau_{1\text{hop}}$ uncertain. Hence, in order to facilitate constructing the performance models, we consider the contention latency from another angle. Since network contention occurs only when multiple communications issued by different processor nodes appear simultaneously in the on-chip network, we introduce an abstract concept: *equivalent serial communication*. The equivalent serial communications are sequential when the program is running so that network contention does not exist at all. To a certain extent, the number of equivalent serial communication (ω) reflects the network contention. Equivalent serial communication is discussed in detail in Step 3 below.

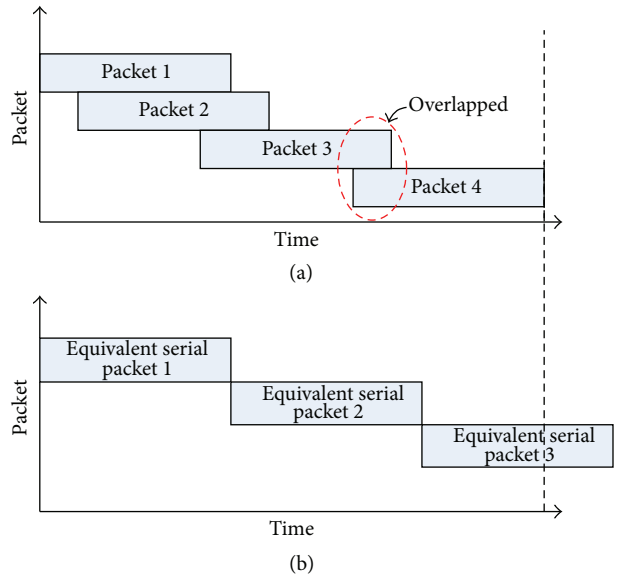

Figure 3: Packets in a communication issued by a processor node.

In the next, we use three steps to establish the communication latency model.

Step 1 (calculating the time of transmitting a packet). With packet switching, the average time of transmitting a packet in the network is

$$\tau_t = H \cdot \tau_{1\text{hop}}, \tag{1}$$

where H reflects the distance and $\tau_{1\text{hop}}$ reflects the architectural latency without contention.

Step 2 (calculating the time of a communication). In general, a communication issued by a processor node contains one or more packets. These packets are launched by the same processor node. Packet transmissions may overlap. In the best case, a packet in a communication is launched one cycle after the preceding packet. A packet transmits in the on-chip network without need of waiting for the completion of its preceding packet transmission. The packet transmissions are overlapping. For the worst case, all packets are transmitted serially; that is, a packet will not be transmitted until the previous one is finished. The overlap among packet transmissions improves the performance by shortening the network communication latency.

To measure the time of a communication, we define an abstract concept: *equivalent serial packet*. The equivalent serial packet is considered to be transmitted sequentially. A communication is abstracted to consist of several equivalent serial packets. As shown in Figure 3(a), assuming that the communication contains four packets, the program behavior determines the concurrent degree of packets' transmission.

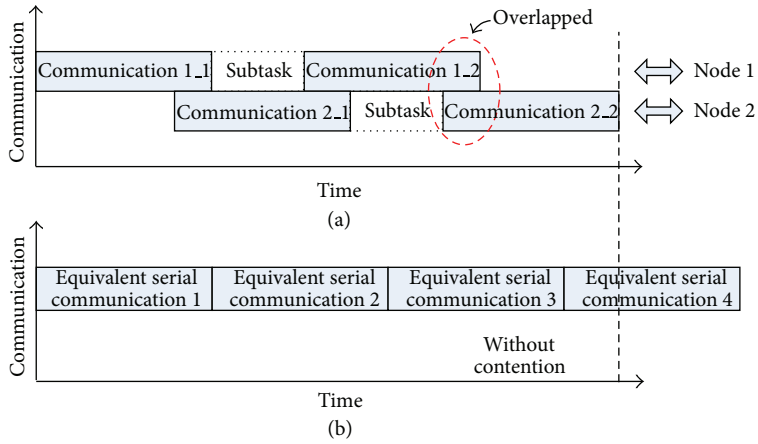

Figure 4: Communications in the entire program.

For example, Packet 1 and Packet 2 are almost fully overlapped, while small portion of Packet 3 and 4 are overlapped. For ease of measuring the communication time, the communication is abstracted to be composed of several equivalent serial packets. In Figure 3(b), the number of equivalent serial packets (γ) is about 2.67, which is less than the packet number: 4. γ meets the inequation below:

$$1 < \gamma \leqslant M. \tag{2}$$

γ describes the concurrent degree of packet transmission in a communication. The ideal best case is that all packets is transmitted concurrently. However, it cannot be reached, because there is only one physical channel from the node to the router. The best case is that packets in a communication are launched one cycle by one cycle, so γ is close to, but not equal to, 1. For the worst case that all packets transmit sequentially, $\gamma = M$. That means the number of equivalent serial packets is equal to the number of real packets (M) in a communication.

From (1) and (2), we can obtain the time of a communication:

$$\tau_c = \gamma \cdot \tau_t = \gamma \cdot H \cdot \tau_{1\text{hop}} \quad (1 < \gamma \leqslant M). \tag{3}$$

Step 3 (calculating the communication overhead of a program). The program is parallelized on N nodes, so the subprogram in each node contains p/N communications. Communications issued by the same node are sequential, because the subprogram is sequentially executed in the processor node. Communications issued by different nodes may exist in the network at the same time. For the best case, the program is fully parallelized. The communication overhead of the entire program is equal to communication latency of the subprogram distributed in each node. For the worst case, communications from different nodes do not overlap one another. The communication overhead of the entire program is equal to the sum of communication latency of each node. In this case, there is no network contention. However, in general, communications are partially overlapped and network contention always exists. Moreover, the existence of multiple communications in the network leads to the occurrence of network contention. The behavior of parallel programs (e.g., "when a communication is generated" and "which node sends or receives packets in the communication") determines the concurrent degree of communications and the network contention latency.

Therefore, in order to quantify the network contention and measure the communication overhead of the entire program, we define an abstract concept: *equivalent serial communication*. The equivalent serial communication is considered to be sequential so that there is no network contention. A program is abstracted to contain several equivalent serial communications. As shown in Figure 4(a), assuming that the program is mapped on two nodes: Node 1 and Node 2. There are four communications. Communication 1_1 and Communication 1_2 are generated by Node 1, while Communication 2_1 and Communication 2_2 are generated by Node 2. Communications generated by different nodes may be overlapped due to the program behavior. For example, Communication 1_2 is overlapped with Communication 2_2. There are network contention between Communication 1_2 and Communication 2_2. The number of equivalent serial communications (ω) is about 3.33, which is less than the communication number: 4. ω meets the inequation below:

$$\frac{p}{N} \leqslant \omega \leqslant \frac{p}{N} \cdot N = p. \tag{4}$$

ω describes the concurrent degree of communications as well as the network contention. The equivalent serial communications are sequential when the program is running so that no network contention occurs. Therefore, the contention latency is removed and fused into the ω when calculating the network communication latency. The network contention and the concurrent degree of communications together determine the value of ω.

(i) If communications are concurrent and they all exist in the same local area resulting in a *hotspot*,

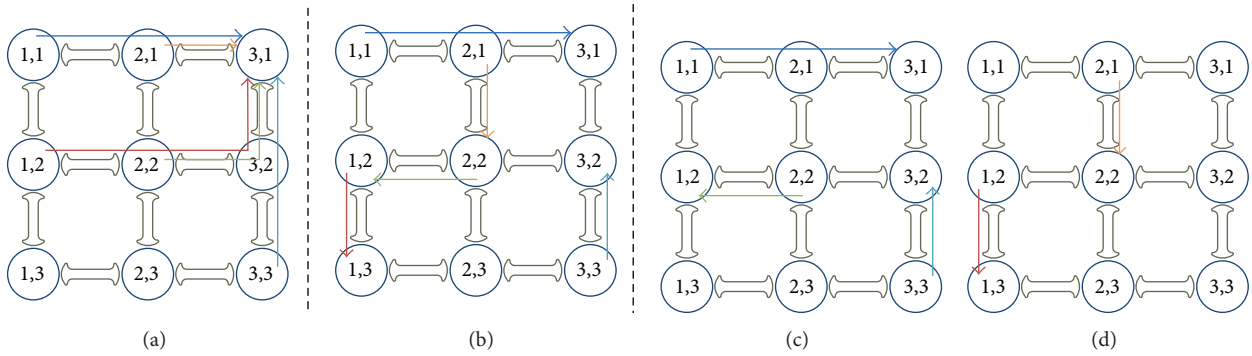

Figure 5: Examples of communications in a 3 × 3 2D-mesh network.

the network contention is heavy. In this case, the total communication time of the program is longer and hence ω is larger, close to $(p/N) \cdot N = p$. For instance, as illustrated in Figure 5(a), Node (1,1), Node (2,1), Node (1,2), Node (2,2), and Node (3,3) communicate with Node (3,1) concurrently. A hotspot is formed near Node (3,1) and network contention is heavy there. Although the five communications are issued concurrently, the network contention serializes them.

(ii) If communications are concurrent and they are uniformly distributed in the entire on-chip network, the network contention becomes light. In this case, the total communication time of the program is shorter and hence ω is smaller, close to p/N. For instance, as shown in Figure 5(b), there are also five communications occurring concurrently in the network. However, they belong to different source nodes and destination nodes and their routing tracks do not overlap, so there is no network contention. Therefore, its ω is smaller than that in Figure 5(a).

(iii) If communications are sequential, although the network contention is not heavy, the total communication time of the program is always long and hence ω is large and close to $(p/N) \cdot N = p$. For instance, as shown in Figure 5(c), Node (1,1) communicates with Node (3,1), Node (2,2) communicates with Node (1,2), and Node (3,3) communicates with Node (3,2). After that, Node (2,1) communicates with Node (2,2) and Node (1,2) communicates with Node (1,3) (see Figure 5(d)). Although there is no network contention, the five communications are not issued concurrently. Therefore, its ω is bigger than that in Figure 5(b).

(iv) For the best case that all nodes are fully concurrent and there is no network contention, the number of equivalent serial communication is equal to the number of real communication in each node ($\omega = p/N$). For the worst case that communications from all nodes occur sequentially, the number of equivalent serial communication is equal to the sum of the number of real communication in each node ($\omega = (p/N) \cdot N = p$).

From (1), (2), and (4), we can calculate the communication overhead of a program running on homogenous OLPCs:

$$T_T = \omega \cdot \tau_c = \omega \cdot \gamma \cdot H \cdot \tau_{1\text{hop}} \quad \begin{pmatrix} 1 < \gamma \leqslant M \\ \dfrac{p}{N} \leqslant \omega \leqslant p \end{pmatrix}. \quad (5)$$

From (5), we can observe that (i) when $\gamma = M$ and $\omega = p$, $T_T = p \cdot M \cdot H \cdot \tau_{1\text{hop}}$ is the maximal communication overhead of the program for the worst case that all packets are transmitted in the network in a sequential way and (ii) when $\gamma \to 1$ and $\omega = p/N$, $T_T \to (p/N) \cdot H \cdot \tau_{1\text{hop}}$ is the minimal communication overhead of the program for the ideal best case that all packets in a communication are transmitted concurrently, all communications from different nodes are concurrent and no network contention occurs and (iii) when the network size is scaled up, H and T_T increases due to the longer communication distance.

Network contention is hard to quantify exactly. The concrete behavior of parallel applications leads to different traffic patterns, packet generation rate, and other factors. These factors influence the network contention. In this section, by introducing two abstract concepts, *equivalent serial packet* and *equivalent serial communication*, we could quantify the network contention and formulate the network communication latency. The equivalent serial packets and equivalent serial communications are sequential so that the network contention does not exist. To a certain extent, the effect of network contention is fused into the number of equivalent serial packet (γ) and the number of equivalent serial communication (ω). With the two extremes of traffic patterns (*Uniform* and *Hotspot* traffic models), we obtain the upper and lower bounds of γ and ω (see Formulas (2) and (4)). The bounds are determined by the number of packets in a communication (M), parallel part of the program (p) and the total processor number (N). M reflects the packet generation rate. Our model offers a feasible way to evaluate the network communication latency of homogenous OLPCs, but here comes a question: how do we determine or estimate

N, p, M, γ, and ω? The network size of OLPCs decides N. Different applications have their own p. Data-parallel applications are scalable and their data are regular. Their programs generally consist of a set of identical subtasks. Analyzing computation and communication behavior of the subtask, we could determine M and estimate γ and ω. Section 5.3 exemplifies the way of estimating γ and ω. Based on the analysis in this subsection, we could have a piece of implication.

Implication 1. Network communication latency has significant influence on the system's performance. The basic three threads to reduce the latency are (1) decreasing the number of communications in the program and the number of packets in a communication, (2) improving the concurrency of communications and packets, and (3) avoiding hotspot traffic. Architects or programmers can try their best to achieve these three threads by optimizing hardware design and application mapping, for instance by offering support for outstanding transactions or caching remote data in the local memory.

4.4. Performance Model. In this subsection, inspired by Amdahl's Law, we establish the performance model for homogenous OLPCs, incorporating the network communication latency. We elaborate the performance model under both *uniform* and *hotspot* traffic patterns. Under the two traffic models, we discuss and analyze the performance's trend, limitation, minimum, and maximum. The impact of network size (N), the ratio of the serial part and the parallel part in a program (α), the number of equivalent serial packets in a communication (γ), and the execution time of a subtask (τ_{nc}) on the performance are also discussed in detail. γ reflects the influence of network contention and congestion, while τ_{nc} reflects the influence of noncommunication/communication ratio.

The same as with Amdahl's Law, we assume that the total problem size is fixed as the number of computing nodes increases. The parallel part in the program is speeded up. The parallel part assigned to each processor node decreases with the increase of the system size. So we can get the performance model as the formula below shows:

$$S = \frac{(s+p)\cdot \tau_{\text{nc}}}{s\cdot \tau_{\text{nc}} + (p/N)\cdot \tau_{\text{nc}} + T_T}. \tag{6}$$

By including (5), we can get

$$S = \frac{(s+p)\cdot \tau_{\text{nc}}}{s\cdot \tau_{\text{nc}} + (p/N)\cdot \tau_{\text{nc}} + \omega\cdot\gamma\cdot H\cdot \tau_{\text{1hop}}} \quad \begin{pmatrix} 1 < \gamma \leqslant M \\ \dfrac{p}{N} \leqslant \omega \leqslant p \end{pmatrix}. \tag{7}$$

The last product item in the denominator describes the communication overhead. If this item is ignored, (6) can be simplified to

$$S = \frac{s+p}{s+p/N} \tag{8}$$

which is Amdahl's Law [6].

The behavior of parallel programs determines the communication patterns, affecting the value of γ and ω. Uniform traffic model is a well-distributed traffic model, while hotspot traffic model is a centralized traffic model. They are two extremes, representing the upper bound and the lower bound of the communication patterns, respectively. Hence, we consider both uniform and hotspot traffic models below to analyze the speedup in detail. Although hotspot traffic has smaller average hop count and hence less minimal latency, hotspot traffic causes much heavier network contention than uniform traffic. For uniform traffic, it has lower network contention and ω is closer to p/N. For hotspot traffic, it has higher network contention and ω is closer to p, because of the serialization effect in the destination node. Therefore, to facilitate the formula transformation and analysis, we consider $\omega = p/N$ for uniform traffic, while $\omega = p$ for hotspot traffic. This assumption is thought to be reasonable without the loss of analyzing the performance trend.

4.4.1. Uniform Traffic Model. Assuming $\omega = p/N$, we can refine (7) as

$$\begin{aligned} S &= \frac{(s+p)\cdot \tau_{\text{nc}}}{s\cdot \tau_{\text{nc}} + (p/N)\cdot \tau_{\text{nc}} + p\cdot\gamma\cdot(2/3)\cdot\left(1/N^{1/2} - 1/N^{3/2}\right)\cdot \tau_{\text{1hop}}} \\ &= \frac{(\alpha+1)\cdot \tau_{\text{nc}}}{(\alpha+1/N)\cdot \tau_{\text{nc}} + (2/3)\cdot\gamma\cdot\left(1/N^{1/2} - 1/N^{3/2}\right)\cdot \tau_{\text{1hop}}}. \end{aligned} \tag{9}$$

Since τ_{1hop} reflects the architectural latency without contention, it is a constant for a given homogenous OLPC architecture. Therefore, The speedup (S) is a quaternion function: $S = S(N, \alpha, \tau_{\text{nc}}, \gamma)$. Its value is determined by N, α, τ_{nc}, and γ. To obtain the variation trend of S, we conduct two steps below.

Step 1 (calculating the speedup's limitation). We have the limitation of S as below:

$$\lim_{N\to\infty} S = \frac{\alpha+1}{\alpha} = 1+\frac{1}{\alpha}. \tag{10}$$

Step 2 (calculating the value of N related to the extreme minimal value of S). Let $\partial S/\partial N = 0$; then, we can get

$$N_{\partial S/\partial N=0} = \frac{6\cdot\gamma^2\cdot\tau_{1\text{hop}}^2 + 9\cdot\tau_{\text{nc}}^2 - 3\cdot\sqrt{12\cdot\gamma^2\cdot\tau_{1\text{hop}}^2\cdot\tau_{\text{nc}}^2 + 9\cdot\tau_{\text{nc}}^4}}{2\cdot\gamma^2\cdot\tau_{1\text{hop}}^2}. \tag{11}$$

Let $\beta = \tau_{\text{nc}}/(\gamma\cdot\tau_{1\text{hop}})$; Formula (11) is refined as

$$N_{\partial S/\partial N=0} = \frac{9\cdot\beta^2 + 6 - \sqrt{9\cdot\beta^2}\cdot\sqrt{9\cdot\beta^2+12}}{2}. \tag{12}$$

From formula (12), we can get

$$\begin{aligned} N_{\partial S/\partial N=0} &> \frac{9\cdot\beta^2+6-\left(9\cdot\beta^2+9\cdot\beta^2+12\right)/2}{2} = 0, \\ N_{\partial S/\partial N=0} &< \frac{9\cdot\beta^2+6-\sqrt{9\cdot\beta^2}\cdot\sqrt{9\cdot\beta^2}}{2} = 3. \end{aligned} \tag{13}$$

The extreme minimal value of S exists; its related N is defined as N_{opt}. Because N is a positive integer, we have

$$N_{\text{opt}} \in \{1,2,3\}. \tag{14}$$

The OLPC hosts at least one processor core, so $N \geqslant 1$. Combining the two steps, we can obtain that

(1) when $N_{\text{opt}} = 1$,

(i) S monotonically increases with the increase of N; parallelization enables the performance improvement; however, S is bounded by $1 + 1/\alpha$ for $N \to \infty$; the ratio of the serial part in a program limits the performance improvement;

(2) when $N_{\text{opt}} = 2$ or $N_{\text{opt}} = 3$,

(i) when $N < N_{\text{opt}}$, S decreases with the increase of N; parallelization degrades the performance rather than improves it, because the negative effect of network communication latency on the performance surpasses the positive effect of cooperation of multiple processor cores on the performance;

(ii) when $N = N_{\text{opt}}$, S reaches its minimum (S_{min});

(iii) when $N > N_{\text{opt}}$, S increases when N is increasing; the positive effect of parallelization surpasses the negative effect of network communication latency, thus improving the performance;

(3) the ratio between the serial part and the parallel part in a program determines the upper limit of S. The limit is inversely proportional to α. It indicates that reducing the serial part or enlarging the parallel part in a program is good for improving the performance limit.

As we can see, S reaches its minimum when N is very small. The OLPC hosts a number of processor cores. Therefore, for a lager range of N, S keeps going up when N increases. To further discuss the effect of N, α, τ_{nc}, and γ on S, Figure 6 shows performance trends of S under uniform traffic model. Without loss of trend analysis, we consider

(a) $N \in \{n \mid 1 \leqslant n \leqslant 256,\ n \in \mathbb{N}\}$; the network size is scaled up from 1 to 256; the increase of the network size makes more processor cores involved;

(b) $\alpha \in \{x \mid 0 \leqslant x \leqslant 1,\ x \in \mathbb{R}\}$; with the increase of α, the serial part takes more proportion in a program; the performance limit $(1 + 1/\alpha)$ becomes less;

(c) $\gamma \in \{1, 16, 256\}$; the number of equivalent serial packets in a communication increases from 1, 16 to 256; more packets lead to larger network communication latency, causing negative effect on the performance;

(d) $\tau_{\text{nc}} \in \{10, 100, 1000\}$; the execution time of a subtask increases from 10, 100 to 1000; increasing noncommunication time can bring positive effect on the performance.

From aforementioned formula transformation and Figure 6, we can have results regarding the performance under uniform traffic model.

(I) The increase of the network size (N) makes more processor cores, exploiting larger parallelism. As shown in Figure 6, as N increases, the speedup (S) firstly decreases and soon reaches its minimum when $N = N_{\text{opt}}$ (this situation is shown in Figure 6(g). It also exists in other subfigures, but it is not obvious, since S is much larger); then, S increases. However, S increases more and more slowly; it is limited by $1+1/\alpha$ finally.

(II) Both the incremental ratio and the limit of S are deeply influenced by α. As shown in all subfigures, as α increases, the incremental ratio of S becomes very low and the limit of S is very small. Even if the network

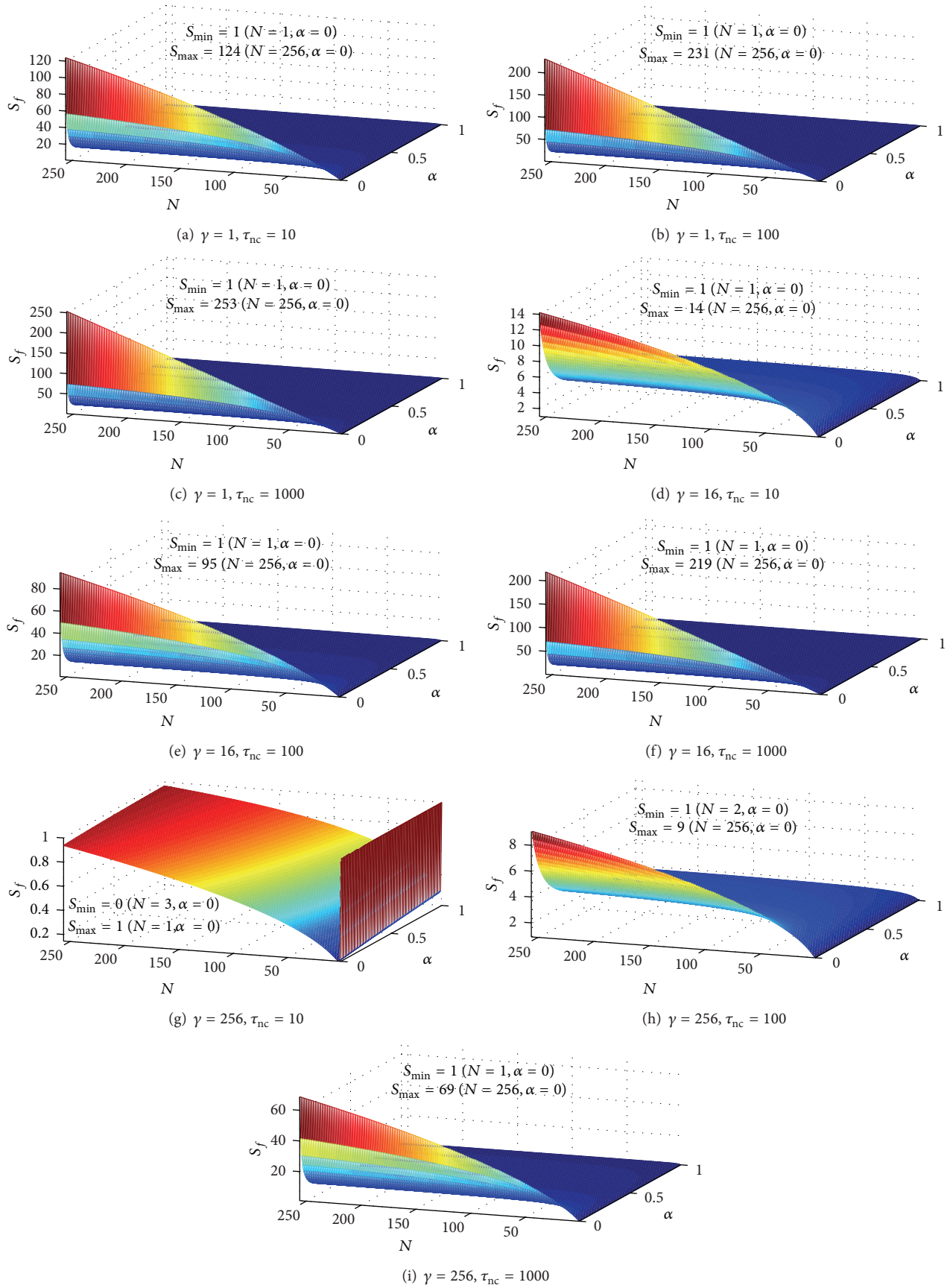

Figure 6: Performance trends under uniform traffic model.

size (N) is scaled up, the performance improvement is very little.

(III) As γ increases, network communication hosts more packets, worsening network congestion or contention and thus generating larger network communication latency. Larger network communication latency brings negative effect on the performance. Frequent network communication and huge latency makes the performance very bad. For instance, for $\tau_{nc} = 10$, $\alpha = 0$, and $N = 256$ (see Figures 6(a), 6(d), and 6(g)), (i) when $\gamma = 1$, S can reach its maximum ($S_{max} = 124$); (ii) when $\gamma = 16$, the maximal speedup becomes small ($S_{max} = 14$); (iii) when $\gamma = 256$, the heavy network communication makes the performance even not improved.

(IV) The increase of τ_{nc} can improve the performance, alleviating and making up the negative effect of network communication latency. For instance, for $\gamma = 16$, $\alpha = 0$, and $N = 256$ (see Figures 6(d), 6(e), and 6(f)), (i) when $\tau_{nc} = 10$, S reaches its maximum ($S_{max} = 14$); (ii) when $\tau_{nc} = 100$, the maximal speedup becomes large ($S_{max} = 95$); (iii) As τ_{nc} rises up to 1000, the maximal speedup ($S_{max} = 219$) is close to the ideal maximal value (256).

4.4.2. Hotspot Traffic Model. Assuming $\omega = p$ and k is odd, we can refine (7) as

$$S = \frac{(s+p)\cdot\tau_{nc}}{s\cdot\tau_{nc} + (p/N)\cdot\tau_{nc} + p\cdot\gamma\cdot\left(\sqrt{N}/2\right)\cdot\tau_{1hop}} = \frac{(\alpha+1)\cdot\tau_{nc}}{(\alpha+1/N)\cdot\tau_{nc} + (1/2)\cdot\gamma\cdot\sqrt{N}\cdot\tau_{1hop}}. \quad (15)$$

The same as with Section 4.4.1, the speedup (S) is also a quaternion function: $S = S(N, \alpha, \tau_{nc}, \gamma)$. Its value is decided by N, α, τ_{nc}, and γ. In (15), when N becomes larger, $(1/N)\cdot\tau_{nc}$ decreases but $(1/2)\cdot\gamma\cdot\sqrt{N}\cdot\tau_{1hop}$ increases, so S may increase or decrease. To obtain the variation trend of S, we also conduct two steps below.

Step 1 (calculating the speedup's limitation). We have the limitation of S as below:

$$\lim_{N\to\infty} S = 0. \quad (16)$$

Step 2 (calculating the value of N related to the extreme maximal value of S). Let $\partial S/\partial N = 0$; then, we can get

$$N_{\partial S/\partial N=0} = \sqrt[3]{\left(\frac{4\cdot\tau_{nc}}{\gamma\cdot\tau_{1hop}}\right)^2} > 0. \quad (17)$$

The extreme maximal value of S exists; its related N_{opt} is obtained by the formula below:

$$N_{opt} = \begin{cases} 1, & \text{when } N_{\partial S/\partial N=0} < 1 \\ \lfloor N_{\partial S/\partial N=0} \rfloor, & \text{when } N_{\partial S/\partial N=0} \geqslant 1, \\ & S(\lfloor N_{\partial S/\partial N=0} \rfloor) \geqslant S(\lceil N_{\partial S/\partial N=0} \rceil) \\ \lceil N_{\partial S/\partial N=0} \rceil, & \text{when } N_{\partial S/\partial N=0} \geqslant 1, \\ & S(\lfloor N_{\partial S/\partial N=0} \rfloor) < S(\lceil N_{\partial S/\partial N=0} \rceil). \end{cases} \quad (18)$$

With Formulas (15) and (17), we can have the extreme maximal value of S:

$$S_{max} = S\left(N_{opt}\right) \simeq S\left(N_{\partial S/\partial N=0}\right) = \frac{\alpha+1}{\alpha + 3\cdot\sqrt[3]{\left(\left(\gamma\cdot\tau_{1hop}\right)/\left(4\cdot\tau_{nc}\right)\right)^2}}. \quad (19)$$

Let $\beta = 3\cdot\sqrt[3]{((\gamma\cdot\tau_{1hop})/(4\cdot\tau_{nc}))^2}$; Formula (19) is refined as

$$S_{max} \simeq \frac{\alpha+1}{\alpha+\beta}. \quad (20)$$

Because N is a positive integer, combining the two steps, we can obtain that

(1) when $N_{opt} = 1$,

(i) S monotonically decreases with the increase of N; parallelization degrades the performance rather than improves it, because the negative effect of network communication latency on the performance surpasses the positive effect of cooperation of multiple processor cores on the performance; S tends to zero when $N \to \infty$;

(2) when $N_{opt} \geqslant 2$,

(i) when $N < N_{opt}$, S increases with the increase of N; within this condition, the network communication latency is not much and parallelization is able to improve the performance;

(ii) when $N = N_{opt}$, S reaches its maximum (S_{max});

(iii) when $N > N_{opt}$, S becomes decreasing when N keeps going up; performance degrades because the network communication latency dominates;

(3) $N_{\partial S/\partial N=0} \sim \tau_{nc}^{2/3}$ and $N_{\partial S/\partial N=0} \sim \gamma^{-2/3}$; when $\tau_{nc} \uparrow$ and $\gamma \downarrow$, $N_{\partial S/\partial N=0} \uparrow$, resulting in $N_{opt} \uparrow$; it indicates that increasing noncommunication time and improving packet concurrency can increase the extreme value of S and the performance improves further covers a larger system size.

To further discuss the effect of N, α, τ_{nc}, and γ on S, Figure 7 shows performance trends of S under uniform traffic model. We consider the values of N, α, τ_{nc}, and γ as the same with Section 4.4.1. From aforementioned formula transformation and Figure 7, we can have results regarding the performance under hotspot traffic model.

(I) Although the increase of the network size (N) could make more processor cores involved to cooperation together so as to seek higher parallel performance, it also induces network communication latency, limiting the performance improvement and even worsen-ing the performance. As shown in Figure 7, as N increases, in some cases (see Figures 7(a), 7(b), 7(c), 7(e), 7(f), and 7(i)), the speedup (S) firstly increases and then becomes decreasing after reaching its maximum; in other cases (see Figures 7(d), 7(g), and 7(h)), it monotonically decreases. For all cases, as N increases, S finally tends to zero.

(II) Both the incremmental/decremental ratio and the maximal value of S are influenced by α. As shown in all subfigures, as α increases, the incremental/decremental ratio becomes very low. With the increase of α, the maximal value of S may increase or decrease: (i) if $1 > \beta$ in Formula (20), S_{max} decreases (see Figures 7(a), 7(b), 7(c), 7(e), 7(f), and 7(i)); (ii) if $1 = \beta$ in Formula (20), $S_{max} \equiv 1$; (iii) if $1 < \beta$ in Formula (20), S_{max} increases (see Figures 7(d), 7(g), and 7(h)).

(III) As γ increases, network communication hosts more packets. Larger network communication latency makes the performance goes bad. The maximal value of S reached by parallelism declines. For instance, for $\tau_{nc} = 1000$ and $\alpha = 0$ (see Figures 7(c), 7(f), and 7(i)), (i) when $\gamma = 1$, S can reach its maximum ($S_{max} = 84$ with $N = 252$); (ii) when $\gamma = 16$, the maximal speedup becomes small ($S_{max} = 13$ with $N = 40$); (iii) when $\gamma = 256$, the heavy network communication makes the speedup soon reach its little maximum ($S_{max} = 2$ with $N = 6$).

(IV) The increase of τ_{nc} can improve the performance, alleviating and making up the negative effect of network communication latency. For instance, for $\gamma = 1$ and $\alpha = 0$ (see Figures 7(a), 7(b), and 7(c)), (i) when $\tau_{nc} = 10$, the maximal value of S is very small ($S_{max} = 4$ with $N = 12$); (ii) when $\tau_{nc} = 100$, the maximal speedup becomes big ($S_{max} = 18$ with $N = 54$); (iii) as τ_{nc} rises up to 1000, the maximal speedup further becomes large ($S_{max} = 84$ with $N = 252$).

In all, performance under hotspot traffic model is worse than that under uniform traffic model.

With the performance analysis in this subsection, we could have the following.

Implication 2. With the uniform traffic model, the communication overhead is modest, assuming that there is limited contention, so the performance can still keep improving. Under the uniform traffic model, the concurrent degree of communications are usually high. Architects or programmers need to pay more attention to improve the concurrent degree of packets in a communication. The performance improvement can benefit more from the improvement of packet concurrency.

Implication 3. With the hotspot traffic model, parallelization cannot always improve the system's performance, when the network communication latency dominates. To alleviate the impact of network communication latency on the performance and hence keep the performance's improvement continuous, designers need to address increasing the noncommunication time and improving packet concurrency.

Implication 4. Exploiting the parallelism of multiple processor cores well is potential to make up the negative effect of network communication latency and even obtains the continuous improvement of the performance. Following this view, architects or programmers need to pay more attention to exploit the parallelism of processor cores.

Implication 5. Besides, increasing the noncommunication time is a viable way to alleviate the negative effect induced by the network communication latency.

5. Experiments and Results

In this section, we apply three real data-parallel applications on our cycle-accurate homogenous OLPC experimental platform to validate and demonstrate the effectiveness of our performance analysis.

5.1. Experimental Platform. Figure 8 shows our homogenous OLPC experimental platform. The platform uses the LEON3 [14] as the processor in each PM node and uses the Nostrum NoC [15] as the on-chip network. Each Processor-Memory (PM) node has a LEON3 processor, an enhanced memory controller plus a local memory. The enhanced memory controller extends the function of LEON3's own memory control module to support memory accesses from/to remote nodes via the network. The LEON3 processor core is a synthesizable VHDL model of a 32-bit processor compatible with the SPARC V8 architecture. The Nostrum NoC is a 2D-mesh packet-switched network with configurable size. Besides, moving one hop in the network takes one cycle.

5.2. Application Examples. We use Wavefront Computation, Vector Norm, and Block Matching Algorithm in Motion Estimation as application examples and perform experiments on various instances of the three applications. Wavefront Computation and Vector Norm are mostly used in wireless communication, computer vision, and image/video processing. And Block Matching Algorithm in Motion Estimation is one of the basic components in image/video processing.

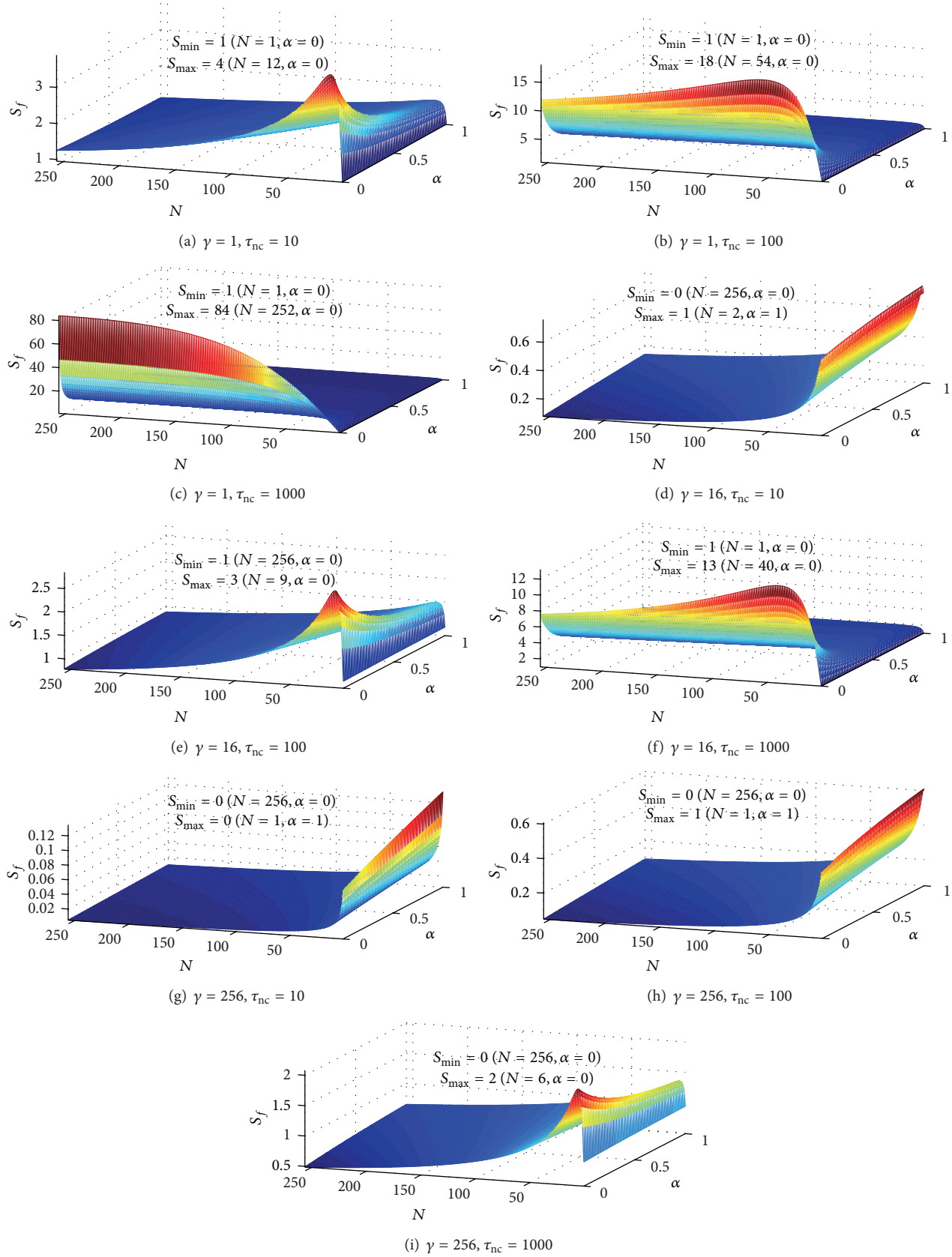

Figure 7: Performance trends under hotspot traffic model.

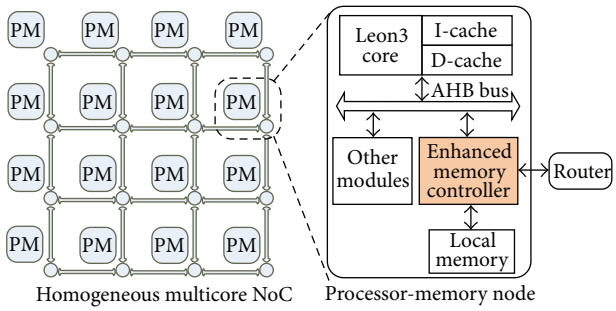

Figure 8: The homogenous OLPC experimental platform.

5.2.1. Wavefront Computation. Wavefront Computations are common in scientific applications. Given a matrix (see Figure 9(a)), the left and top edges of which are all a constant, the computation of each remaining element depends on its neighbors to the left, above, and above-left. If the solution is computed in parallel, the computation at any instant forms a wavefront propagating toward in the solution space. Therefore, this form of computation gets its name as wavefront. We use the same method as [16] to parallelize the Wavefront Computation, the rows of the matrix are assigned to PM nodes in a round-robin fashion (see Figure 9(b)). With this static scheduling policy, to compute an element, only the availability of its above neighbor needs to be checked (synchronized). For instance, PM node 0 computes the elements in row 1. PM node 1 cannot compute the elements in row 2 until the corresponding elements in row 1 has been figured out by PM node 0. After finishing the computation in row 1, PM node 0 goes on to compute the elements in row 3 according to the round-robin scheduling policy. In our experiment, we conduct various instances of Wavefront Computation described below.

(1) Two ways of data storing are realized to reflect the two traffic models. One is "*Uniform*" meaning that the matrix data are uniformly distributed over all nodes. The other is "*Hotspot*" meaning that the matrix data are only located in the central node.

(2) Both integer matrix and floating point matrix are implemented to vary the noncommunication time: τ_{nc}. For the same problem size and algorithm, floating point computation needs more time than integer computation and hence has bigger τ_{nc}.

(3) The Wavefront Computation conducts a matrix with the size of 256×256, on the homogenous OLPC with the network size varying from 1×1 (1), 1×2 (2), 2×2 (4), 2×4 (8), 4×4 (16), 4×8 (32), 8×8 (64), 8×16 (128), to 16×16 (256). The total problem size is fixed and the problem size assigned to each node varies from 256 rows, 128 rows, 64 rows, 32 rows, 16 rows, 8 rows, 4 rows, 2 rows to 1 row.

5.2.2. Vector Norm. Vector Norm is used to compute the magnitude (length) of the vector. Figure 10(a) shows the formula of Vector Norm. When $p = 2$, the Vector Norm is also called L^2-*Norm* or *Euclidean Norm*, which is common in operations of 2D/3D computer graphics. In the paper, we choose to parallelize and compute L^2-*Norm*. Figure 10(b) illustrates the parallelization of L^2-*Norm* on our OLPC platform. Different from Matrix Multiplication and Wavefront Computation, Vector Norm only can be partially parallelized. Its computation contains two steps. Step 1 is parallel. In Step 1, PM nodes are responsible for computing the square (t_i) of x_i ($i = 1, \ldots, n$). x_i are assigned to PM nodes in a round-robin fashion. Step 2 is sequential. In Step 2, a central PM node takes charge of computing the square root of the sum of all t_i. For instance, as shown in Figure 10(b), there are two PM nodes computing the L^2-*Norm* of a vector with four elements. In Step 1, PM node 0 computes the square (t_1) of x_1, while PM node 1 computes the square (t_2) of x_2. After finishing the computation of t_1, PM node 0 goes on to compute the square (t_3) of x_3 according to the round-robin scheduling policy. In Step 2, PM node 1 (the central PM node) computes $\sqrt{t_1 + t_2 + t_3 + t_4}$. In our experiment, we apply various instances of L^2-*Norm* described below.

(1) Two ways of data storing are realized to reflect the two traffic models. One is "*Uniform*" meaning that the data in Step 1 are uniformly distributed over all nodes. The other is "*Hotspot*" meaning that all data in both Steps 1 and 2 are only located in the central node.

(2) Both integer data type and floating point data type are implemented to vary the noncommunication time: τ_{nc}. For the same problem size and algorithm, floating point computation needs more time than integer computation and hence has bigger τ_{nc}.

(3) The L^2-*Norm* conducts a vector with 1024 elements, on the homogenous OLPC with the network size varying from 1×1 (1), 1×2 (2), 2×2 (4), 2×4 (8), 4×4 (16), 4×8 (32), 8×8 (64), 8×16 (128), to 16×16 (256). The total problem size is fixed and the problem size assigned to each node varies from 1024 elements, 512 elements, 256 elements, 128 elements, 64 elements, 32 elements, 16 elements, 8 elements, to 4 elements.

5.2.3. Block Matching Algorithm in Motion Estimation. Motion Estimation is one of important parts in H.264/AVC standard, which addresses obtaining high coding efficiency and good picture quality [17]. It is of importance to find the best Motion Vector in Motion Estimation. The Block Matching Algorithm in Motion Estimation aims at looking for the best matching block with the best Motion Vector in Reference Frame. Figure 11(a) illustrates the Block Matching Algorithm. As shown in the figure, there is a Current Block (**C**) in the Current Frame. For a Reference Frame, the Block Matching Algorithm first predicts a Search Center (**SC**) according to the position of the Current Block (**C**). Then, it exhaustively checks all search points (i.e., candidate Reference Blocks, e.g., **R**) in the Search Window (**SW**) of the Reference Frame to find the best matching block ($\mathbf{R}_{opt}$) with the best Motion Vector (**MV**). The position of the Search Window (**SW**) is decided by the Search Center (**SC**), while its size is decided by the Search Range (**SR**). It is obvious that larger Search

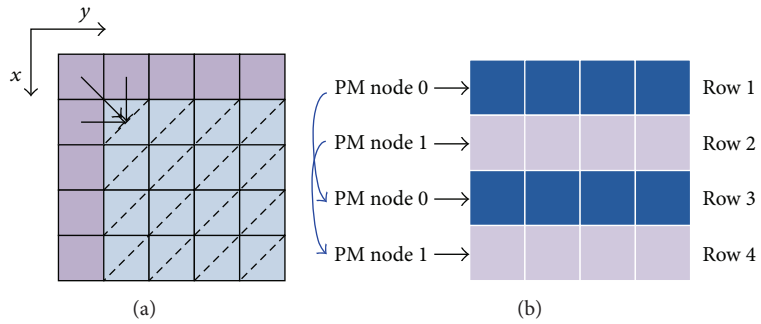

FIGURE 9: (a) Wavefront Computation; (b) its parallelization.

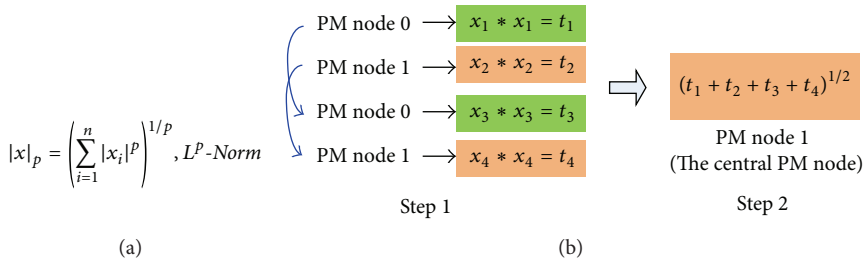

FIGURE 10: (a) Vector Norm and (b) its parallelization.

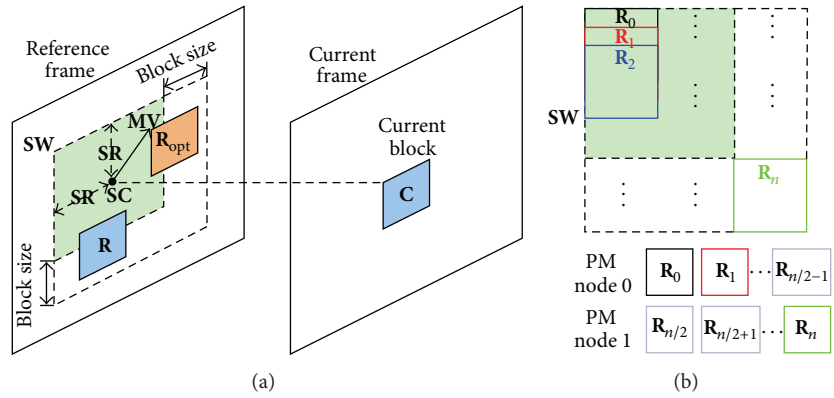

FIGURE 11: (a) Block Matching Algorithm in Motion Estimation and (b) its parallelization.

Window (**SW**) leads to more accurate prediction of the best matching block with the best Motion Vector but consumes more amount of computation time. Figure 11(b) shows how the Block Matching Algorithm is parallelized on our OLPC platform. We uniformly assign candidate Reference Blocks ($\mathbf{R}_i$) into each PM node so that each PM node handles the same number of candidate Reference Blocks. For instance, assume that there are n search points in the Search Window (**SW**) and two PM nodes take charge of obtaining the best matching block. The PM node 0 is responsible for comparing $\mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_{n/2-1}$ with the Current Block (**C**), while PM node 1 takes charge of comparing $\mathbf{R}_{n/2}, \mathbf{R}_{n/2+1}, \ldots, \mathbf{R}_n$ with the Current Block (**C**). In our experiment, we perform a various instances described below.

(1) We also realize two ways of data storing to reflect the two traffic models. One is "*Uniform*" meaning that the candidate reference blocks are uniformly distributed over all nodes. The other is "*Hotspot*" meaning that all candidate reference blocks are located in the central node.

(2) Only integer data type is considered, since the data in image processing are "integer."

(3) We conduct a Search Window with the size of 128 × 128 (i.e., 16384 candidate reference blocks), on the homogenous OLPC with the network size varying from 1 × 1 (1), 1 × 2 (2), 2 × 2 (4), 2 × 4 (8), 4 × 4 (16), 4 × 8 (32), 8 × 8 (64), 8 × 16 (128), to 16 × 16 (256).

The total problem size is fixed and the problem size assigned to each node varies from 16384, 8192, 4096, 2048, 1024, 512, 256, 128, to 64 reference blocks.

5.3. Theoretical Speedup Estimation. To compare our theoretical analysis with the real simulation results, we first estimate the theoretical speedups of the three applications.

5.3.1. Wavefront Computation

(1) The program of Wavefront Computation can be fully parallelized, thus $s = 0$.

(2) The subtask on each node is $E(i, j) = E(i-1, j-1) * E(i-1, j-1) - E(i, j-1) * E(i-1, j)$. Here, $E(i, j)$ represents the current element in the matrix in Figure 9, while $E(i-1, j-1)$, $E(i, j-1)$ and $E(i-1, j)$ are $E(i, j)$'s neighboring element to the above-left, left and above, respectively. The time of such subtask (including computation and local memory reference) is collected in our experiment: $\tau_{nc} = 176$ clock cycles for integer data type; $\tau_{nc} = 2032$ clock cycles for floating point data type.

(3) For "*Uniform*" data storing, the elements computed by a PM node are located on the local memory of that PM node. Hence, $E(i, j)$ and $E(i, j-1)$ are local, while $E(i-1, j-1)$ and $E(i-1, j)$ are remote. There are two packets ($M = 2$) transmission in a communication and we assume that $\gamma = 1.5$ considering packet concurrency. For "*Hotspot*" data storing, all elements are located on the central node. Hence, there are four packet transmissions ($M = 4$) in a communication. Considering packet transmissions are overlapped, we assume that $\gamma = 2$.

5.3.2. Vector Norm

(1) The program of Vector Norm is partially parallelized. The serial part consumes much time.

(2) Step 1 is parallel. In Step 1, the subtask on each node is $t_i = x_i \times x_i$. The time of such subtask (including computation and local memory reference) is collected in our experiment: $\tau_{nc} = 110$ clock cycles for integer data type; $\tau_{nc} = 1270$ clock cycles for floating point data type. Because the vector contains 1024 elements, $p = 1024$. Step 2 is sequential. in our experiment, the computation of $\sqrt{t_1 + t_2 + \cdots + t_{1024}}$ takes 32700 cycles for integer data type and 337560 cycles for floating point data type. So $s = 32700/110 \approx 297$ for integer data type and $s = 337560/1270 \approx 266$ for floating point data type.

(3) For "*Uniform*" data storing, x_i in Step 1 used by a PM node are located on the local memory of that PM node. t_i is stored in the central PM node. Hence, there is one packet ($M = 1$) transmission in a communication and we assume that $\gamma = 1$. For "*Hotspot*" data storing, all data are located on the central node. Hence, there are two packet transmissions ($M = 2$) in a communication. Considering packet transmissions are overlapped, we assume that $\gamma = 1.5$.

5.3.3. Block Matching Algorithm in Motion Estimation

(1) The Reference Frame has been computed and stored in on-chip local memories in the last Motion Estimation. In current Motion Estimation, the "Block Matching" processing starts until the Current Block in the Current Frame is transferred from the off-chip DRAM into the on-chip memory. The elapsed time of transferring the Current Block from the off-chip DRAM memory into the on-chip memory is the serial part of the Block Matching Algorithm. In our OLPC platform, the central PM node features an External Memory Interface connecting with the off-chip DRAM. The External Memory Interface reads a datum from the DRAM in 20 cycles and the size of the Current Block is 16×16. Hence, for "*Hotspot*" data storing that all data are stored in the central PM node, the data transfer takes 5120 ($=16 \times 16 \times 20$) cycles. For "Uniform" data storing that data are uniformly stored in each PM node, the Current Block is transferred from the DRAM to the External Memory Interface and routed to all PM nodes in a broadcast way, so the time of the Current Block's transfer is $5120 + N/2$ cycles (a packet from the central node to the corner one takes $N/2$ hops), approximately equal to 5120 cycles. The subtask on each node is the comparison of the Current Block and a candidate Reference Block, consuming 7680 cycles. Therefore, the problem size is 128×128, so the parallel part takes 125829120 ($=7680 \times 128 \times 128$) cycles. $s = 5120/(5120 + 125829120) \approx 0.00\%$, and $p = 1 - s \approx 100.00\%$.

(2) The subtask on each node is the comparison of the Current Block and a candidate Reference Block. The time of such subtask (including computation and local memory reference) is collected in our experiment: $\tau_{nc} = 7680$ cycles.

(3) For "*Uniform*" data storing, the Current Block and the candidate blocks are located in each PM node, so there is no network communication and $\gamma = 0$. For "*Hotspot*" data storing, the Current Block and the candidate blocks are in the central PM node, Hence, there are 512 ($=16 \times 16 \times 2$) packet transmissions ($M = 512$) in a communication. Considering that such many packets are routed to the central node, the network contention is extremely heavy and we assume that $\gamma = 512$.

Then, using the Formula (9) and (15) estimates the theoretical speedups of the three applications.

5.4. Simulation Results. The real speedups of the three applications are calculated based on the simulation results on our homogenous OLPC experimental platform (because the sequential part in the program of Vector Norm dominates, the performance improvement is limited).

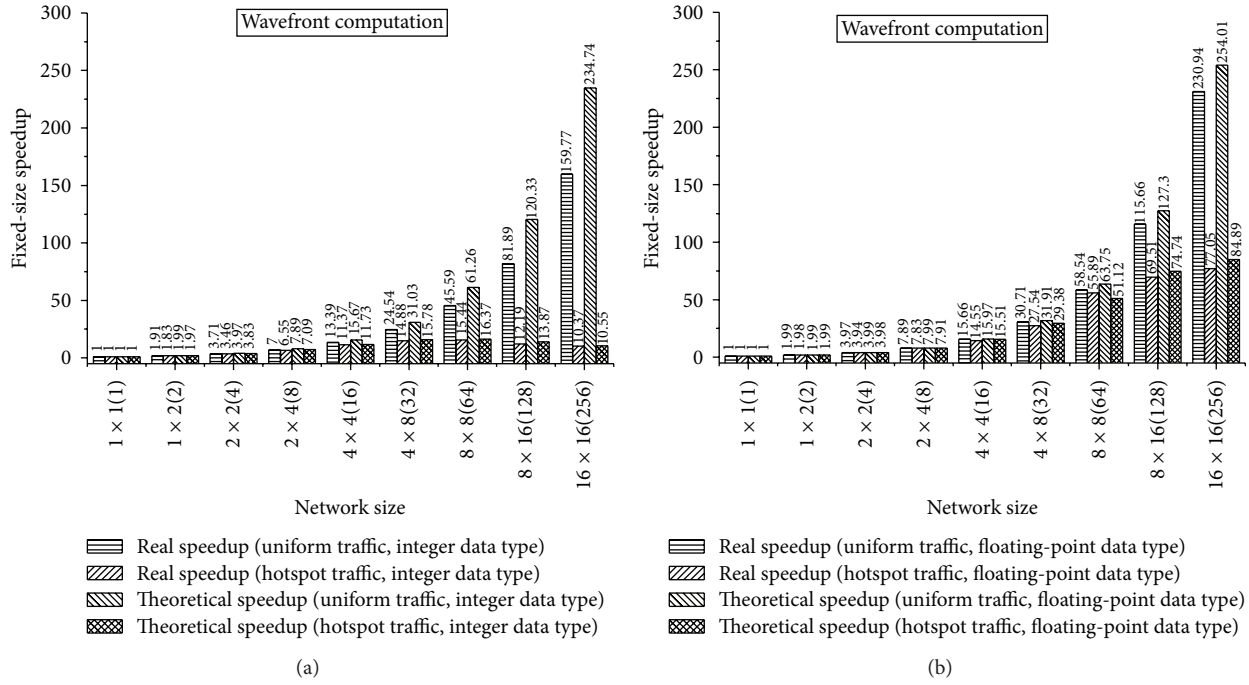

Figure 12: Effect of traffic models: Wavefront Computation with (a) integer data type and (b) floating-point data type.

5.5. Analysis and Discussion

5.5.1. Effect of Network Size. The effect of network size on the performance reflects the scalability of homogenous OLPCs. Figures 12, 13, 14, 15, and 16 plot the real and theoretical speedups versus the size of the homogenous OLPC from 1 × 1 (1), 1 × 2 (2), 2 × 2 (4), 2 × 4 (8), 4 × 4 (16), 4 × 8 (32), 8 × 8 (64), 8 × 16 (128), to 16 × 16 (256). From the six figures, we can see that (i) the theoretical speedups has the same trend with the real speedups; (ii) for uniform traffic model, the speedup usually increases when the network size is scaled up; and (iii) for hotspot traffic model, the speedup reaches its maximum when the network size is scaled up to a certain size and becomes decreasing when the network size goes on increasing.

5.5.2. Effect of Traffic Models. Figure 12 shows the effect of traffic models on the real and theoretical speedups of both integer and floating-point Wavefront Computation, Figure 13 shows the effect of traffic models on the real and theoretical speedups of both integer and floating-point Vector Norm, and Figure 14 shows the effect of traffic models on the real and theoretical speedups of Block Matching Algorithm in Motion Estimation.

(i) For uniform traffic model, consistent with the theoretical speedup performance model, the real speedup increases as the network size is scaled up, no matter the data type is integer or floating-point. This is because the contention latency induced by uniform traffic is not enough to kill the performance improvement introduced by the parallelization. However, it can slow down the performance improvement.

(ii) Because a hotspot traffic model incurs heavy contention latency, the speedup increases when the network size is small but begins decreasing when the network size is scaled up to a certain finite value. Using (17), we can calculate the value of network size (N) for the maximal speedup. (i) For Wavefront Computation with integer data type, $N_{opt} \approx 50$, so Figure 12(a) shows that both the theoretical and the real speedups go up from 1 × 1 (1) to 4 × 8 (32), the speedups on 8 × 8 (64) are approximately equal to the speedups on 4 × 8 (32), and the speedups turn to fall as the network size goes on increasing to 16 × 16 (256). (ii) For Wavefront Computation with floating-point data type, $N_{opt} \approx 255$, so Figure 12(b) shows both the theoretical and the real speedups ascend when the network size is from 1 × 1 (1) to 16 × 16 (256). (iii) For Vector Norm with integer data type, $N_{opt} \approx 44$, so Figure 13(a) shows that both the theoretical and the real speedups go up from 1 × 1 (1) to 4 × 8 (32), the speedups on 8 × 8 (64) are approximately equal to the speedups on 4 × 8 (32), and the speedups fall when the network size goes on increasing to 16 × 16 (256). (iv) For Vector Norm with floating-point data type, $N_{opt} \approx 226$, so Figure 13(b) shows both the theoretical and the real speedups ascend when the network size is from 1 × 1 (1) to 16 × 16 (256). (v) For Block Matching Algorithm, $N_{opt} \approx 15$, so Figure 14 shows that both the theoretical and the real speedups go up from

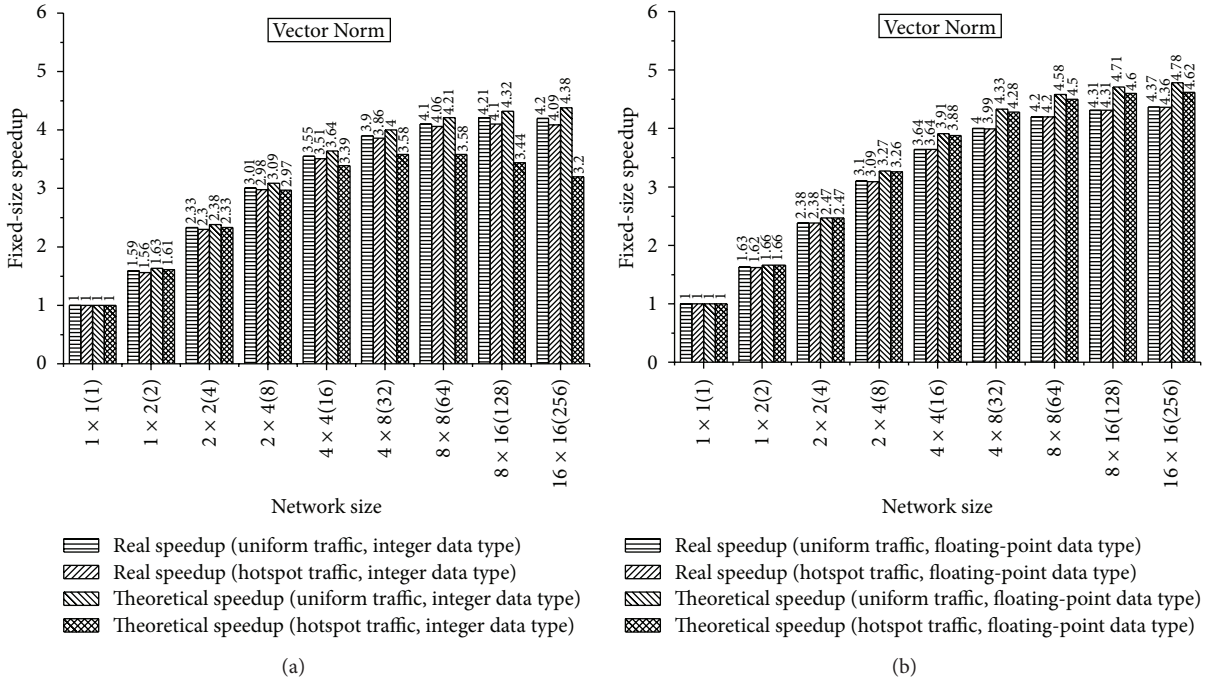

Figure 13: Effect of traffic models: Vector Norm with (a) integer data type and (b) floating-point data type.

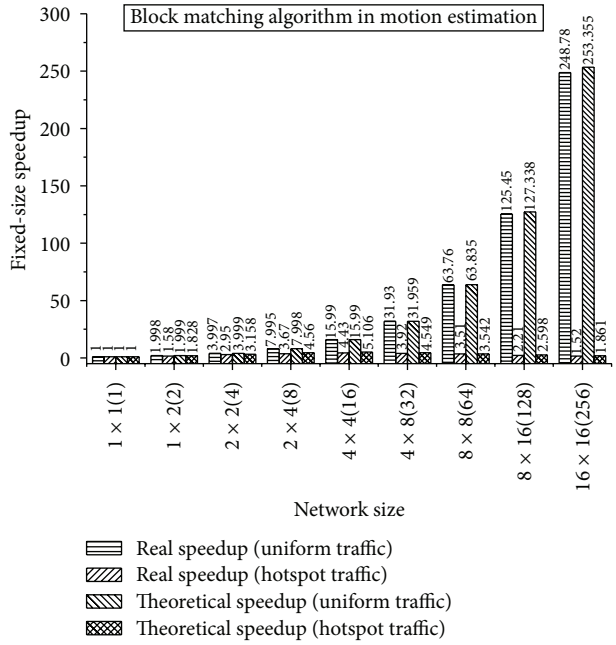

Figure 14: Effect of traffic models: Block Matching Algorithm in Motion Estimation.

1×1 (1) to 4×4 (16) but fall when the network size is from 4×8 (32) to 16×16 (256). From Figure 14, we can see that the speedup under hotspot traffic model is very small, because the Block Matching Algorithm in Motion Estimation makes a large number of memory accesses flow towards the central PM node and hence results in huge network contention.

(iii) Because hotspot traffic model consumes much more network contention latency than uniform traffic model, the speedup with hotspot traffic model is smaller than that with uniform traffic model for the same network size. The difference becomes larger when the network size is increasing.

5.5.3. Effect of Noncommunication/Communication Ratio. Figure 15 shows the effect of noncommunication/ communication ratio on the real and theoretical speedups of Wavefront Computation under both uniform and hotspot traffic models, and Figure 16 shows the effect of noncommunication/communication ratio on the real and theoretical speedups of Vector Norm under both uniform and hotspot traffic models.

(i) For the same network factors, the theoretical and real speedups for the floating point data type is higher than those for the integer data type. This is as expected because when the noncommunication time increases, the portion of communication latency becomes less significant, thus achieving higher performance.

(ii) For hotspot traffic model, the increase of noncommunication/communication ratio shifts the optimal network size (N_{opt}) to a larger value. For integer data type, $N_{opt} \approx 50$ for Wavefront Computation and $N_{opt} \approx 44$ for Vector Norm. For floating-point data type, $N_{opt} \approx 255$ for Wavefront Computation and $N_{opt} \approx 226$ for Vector Norm.

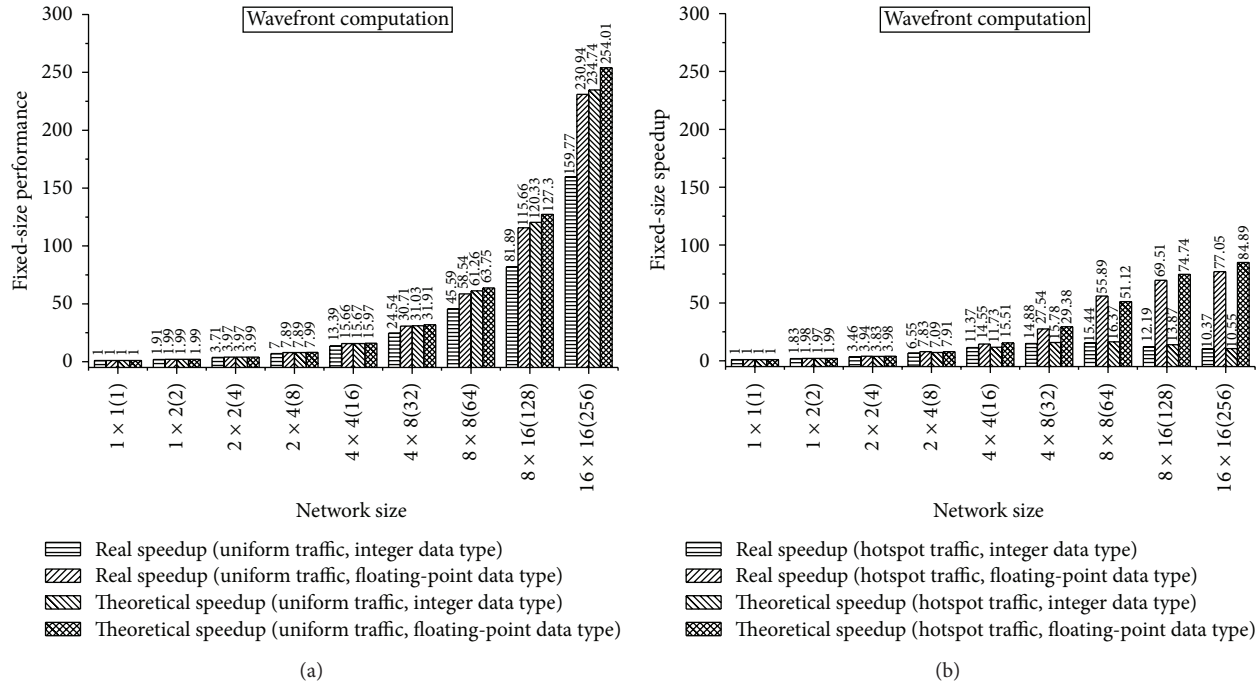

Figure 15: Effect of noncommunication/communication ratio: Wavefront Computation with (a) uniform traffic model and (b) hotspot traffic model.

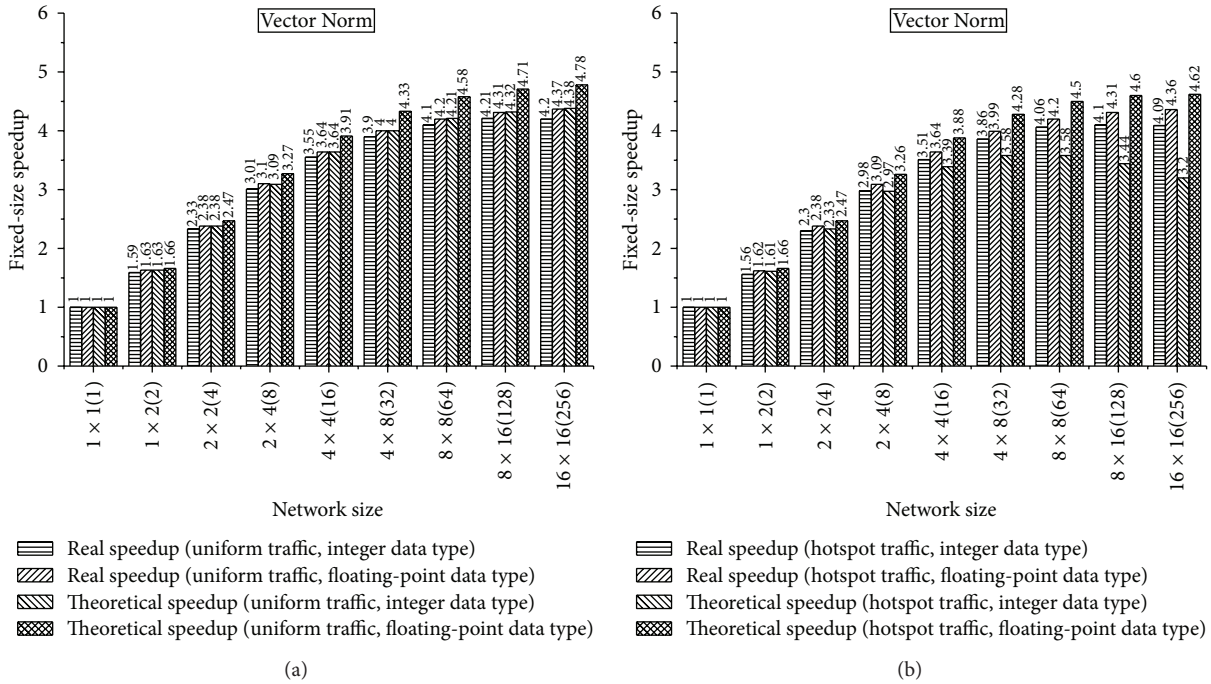

Figure 16: Effect of noncommunication/communication ratio: Vector Norm with (a) uniform traffic model and (b) hotspot traffic model.

6. Applicability and Limitation

6.1. Applicability. The target architectures and applications of our study are homogenous OLPCs and data-parallel applications, respectively. Homogenous OLPCs are such an on-chip computing platform that have a number of computing cores that run in and an on-chip network that provides concurrent pipelined communication, and data-parallel applications represent a wide range of applications whose data sets can be partitioned in parallel

into data subsets handled individually by the same program running in different processor cores. Scalability is the common characteristic of both. Considering that homogenous OLPCs and data-parallel applications match each other very well in nature and hence data-parallel applications can obtain good speedup in homogenous OLPCs, the performance model proposed by the paper is applicable to homogenous OLPCs for data-parallel applications.

The performance model is general for homogenous OLPCs and a variety of data-parallel applications. For any particular application, a customized many-core platform such as application-specific architecture and hardware accelerator will be superior, but a NoC-based homogenous OLPC would be better than such a custom-designed hardware architecture when a variety of data-parallel applications share the same OLPC. The custom-designed many-core platform is specific so as not to be in the range of the general homogenous OLPCs. GPU (Graphic Processing Unit) is such kind of hardware accelerator for graphic processing as the name suggests. Although GPGPU (General-Purpose GPU) exhibits generality to some extent by providing programmability in its GPUs, it is still specific because the programmable GPU adopts a special structure for accelerating the graphic processing applications and the interconnections in GPGPU is special for such as stream processing and data shuffling that are common in graphic processing. Therefore, GPGPU is not in the range of homogenous OLPCs. Besides data-parallel applications, there exist other applications that do not have the scalability feature, so the proposed model is not applicable to those applications' performance analysis.

6.2. Limitation. The proposed performance model is not suitable for many-core platforms in specific application areas and the applications without the characteristic of scalability. The purpose of the model is to offer a general but workable way to estimate and evaluate the performance of homogenous OLPCs for data-parallel applications. Because the network communication latency and the ways of storing data of data-parallel applications are two of the most significant factors affecting the performance of homogenous OLPCs, when we establish the speedup performance model, the network communication latency and the data storing ways are stressed out and modeled in detail. Therefore, the processor behavior such as cache hierarchy and cache miss is not considered. We assume that all of the data are moved from the external memory to the appointed on-chip memory regions in different nodes before the system handles the data and the performance is measured from the time when the system begins handling the data, even if the data is continuously fed from the external memory, part of the latency can be hidden in the process of data handling, and we emphasize analyzing the effect of the data storing ways, so the model does not describe the situation that data are moved from the external memory.

7. Conclusion

Understanding the speedup potential that homogenous OLPC computing platforms can offer is a fundamental question to continually pursuing higher performance. This paper has focused on analyzing the performance of homogeneous OLPCs for data-parallel applications. Because the enhancement of application performance in OLPCs may be restricted by the increasing network communication latency even though the number of cores increases, one main issue for the analysis is to properly capture the network communication. We first detailed a network communication latency model by proposing two abstract concepts (*equivalent serial packet* and *equivalent serial communication*). Then, based on the network communication latency model, we have proposed the performance model. By considering the uniform and hotspot traffic models, the performance model has two detailed forms to reflect the distributed way and the centralized way of storing data of data-parallel applications. Essentially, the performance model revisits Amdahl's Law under the context of homogenous OLPCs. Theoretic analysis and real application experiments demonstrate that our model provides a feasible way to estimate and evaluate the performance of data-parallel applications onto homogenous OLPCs.

In the future, we plan to extend the performance model by considering the cache hierarchy and cache miss and the external memory access. Another direction is to emphasize studying the effect of topologies and communication protocols on the performance models of homogenous OLPCs.

Notations

k: Number of nodes in each dimension
N: The number of processor nodes, $N = k^2$
s: The number of subtasks in the serial part of a program
p: The number of subtasks or communications in the parallel part of a program
α: The ratio between the serial part and the parallel part in a program, $\alpha = s/p$
τ_c: The time of a communication
τ_{nc}: The execution time of a subtask, that is, noncommunication time
H: Average hop count of transmitting a packet
τ_{1hop}: The time of transmitting a packet in one hop
τ_t: Average time of transmitting a packet in the network
M: The number of packets in a communication
γ: The number of equivalent serial packets in a communication
ω: The number of equivalent serial communications in a program
T_T: The communication overhead of a program on OLPCs
S: Speedup
S_{max}: Maximal speedup
S_{min}: Minimal speedup.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The research is partially supported by the Hunan Natural Science Foundation of China (no. 2015JJ3017), the Doctoral Program of the Ministry of Education in China (no. 20134307120034), and the National Natural Science Foundation of China (no. 61402500).

References

[1] S. Borkar, "Thousand core chips—a technology perspective," in *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC '07)*, pp. 746–749, June 2007.

[2] M. Horowitz and W. Dally, "How scaling will change processor architecture," in *Proceedings of the IEEE International Solid-State Circuits Conference, Digest of Technical Papers (ISSCC '04)*, vol. 1, pp. 132–133, February 2004.

[3] A. Jantsch and H. Tenhunen, *Networks on Chip*, Kluwer Academic Publishers, New York, NY, USA, 2003.

[4] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, pp. 1–51, 2006.

[5] J. D. Owens, W. J. Dally, R. Ho, D. N. Jayashima, S. W. Keckler, and L.-S. Peh, "Research challenges for on-chip interconnection networks," *IEEE Micro*, vol. 27, no. 5, pp. 96–108, 2007.

[6] G. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the American Federation of Information Processing Societies Conference*, pp. 383–385, Atlantic City, NJ, USA, April 1967.

[7] X. Li and M. Malek, "Analysis of speedup and communication/computation ratio in multiprocessor systems," in *Proceedings of the Real-Time Systems Symposium*, pp. 282–288, 1988.

[8] J. Paul, "Amdahl's law revisited for single chip systems," *International Journal of Parallel Programming*, vol. 35, no. 2, pp. 101–123, 2007.

[9] S. Cho and R. G. Melhem, "Corollaries to Amdahl's law for energy," *Computer Architecture Letters*, vol. 7, no. 1, pp. 25–28, 2008.

[10] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[11] G. Loh, "The cost of uncore in throughput-oriented manycore processors," in *Proceedings of the Workshop on Architectures and Languages for Throughput Applications (ALTA '08)*, Beijing, China, June 2008.

[12] 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 Transactions on Computers*, vol. 54, no. 8, pp. 1025–1040, 2005.

[13] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004.

[14] Gaisler, "Leon3 processor," http://www.gaisler.com/doc/Leon3%20Grlib%20folder.pdf.

[15] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum backbone—a communication protocol stack for networks on chip," in *Proceedings of the 17th International Conference on VLSI Design*, pp. 693–696, January 2004.

[16] W. Zhu, V. C. Sreedhar, Z. Hu, and G. R. Gao, "Synchronization state buffer: supporting efficient fine-grain synchronization on many-core architectures," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*, pp. 35–45, June 2007.

[17] ITU, "Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264 — ISO/IEC 14496-10 AVC)," 2003.