

A Synthetic Task Model for HPC-Grade Optical Network Performance Evaluation

Sébastien Rumley¹, Lisa Pinals¹, Gilbert Hendry², and Keren Bergman¹

¹*Department of Electrical Engineering, Columbia University, New York, USA*

²*Sandia National Laboratories, Livermore, CA, USA*

Abstract— With vastly increasing system parallelism, energy efficient data movement has emerged as one of the key challenges in High Performance Computing (HPC). Silicon photonics uniquely offers the potential for creating system-wide optical interconnection networks with extremely high bandwidth and energy efficiency. However, to fully reap the benefits of optical data movement, the interconnect design must go beyond a simple wire replacement to include a fully networked architecture. Furthermore, future optically interconnected systems must support a wider range of HPC applications, in particular, those with irregular behavior. Simulation is an essential tool in the design space exploration of future optical network-enabled systems for various applications. However, simulation requires appropriate models capturing the relevant system-interconnection network-application interactions as well as the photonic physical layer integrity. An important goal is to keep the simulation as simple as possible to enable a wide design space exploration. In this paper, we propose a traffic generation model that captures the interactions and dependencies between computation and communication for a given application. Our proposed model remains tractable enough to be implemented on top of detailed photonic network simulators as well as general enough to analyze different components of the network under investigation.

Keywords—Optical interconnects, performance evaluation, high-performance computing.

I. INTRODUCTION

In recent years, parallelization has emerged as the main enabler of High Performance Computing (HPC) speedup [1]. However, rising levels of parallelization induce a drastic increase in the amount of data movement. For this reason, interconnects linking multiple computing elements have become critical components of the overall HPC system. Many novel architectures have been proposed [2][3] in response to this problem. Typical modern HPC interconnects span tens of meters and must sustain Terabytes per second of aggregated bandwidth. Given these requirements, photonics is a promising solution due to its wide offered bandwidth and limited signal attenuation over long distances. Although only a small portion of the links are optical in current super-computers [4], this proportion is likely to increase in the upcoming years as technological advances are leading to cheaper and more efficient optical links. Moreover, alternatives to optics will likely be unable to sustain the future bandwidth requirements [5].

In addition to increasing the proportion of optical links, there is a great incentive to perform switching operations optically to reduce the transfers between the optical and

electronic domains. If optical switching is performed on Wavelength Division Multiplexed (WDM) data, a common technique for optical transmission, the switching of hundreds of Gb/s can be performed in one single operation. The integration of optical switching in data-center type interconnects with Micro Electro Mechanical Systems (MEMS) acting as micro mirrors is proposed in references [6] and [7]. Emerging technologies, such as Silicon Photonics, have also been proposed to perform optical switching [8].

Optics provides wide bandwidth, quasi-distance independence, and fast, low power switching. Furthermore optics can be exploited to perform unconventional networking operations, such as multicast communications [9]. However, no practical optical memories or buffers have been demonstrated to date, presenting a significant challenge for network design. Also, although some optical devices can be operated at sub-nanosecond speeds (i.e. several or even tens of Ghz), others demonstrate much slower reaction times. To either circumvent or exploit these peculiarities, hardware optical interconnect designs typically must be complemented by custom network control mechanisms and protocols. One common function of these control mechanisms and protocols is to shift all the buffering on the optical network edge (where it is achieved by electronic buffers)[10]. These obstacles complicate the design process of a hybrid electrical/optical interconnect. To make the design process even more difficult to define, the design objectives vary with the HPC system target application(s). Some applications primarily require wide bandwidth, while others require fast message passing mechanisms. Other application may have even more specific requirements, such as barrier synchronization [11].

In this context, the interconnect design is not a direct process, but requires a sequence of iterations across multiple platforms, each with a different target level of abstraction. Devices assembled in optical sub-systems are progressively translated into network capabilities. Concurrently, the application requirements can be analyzed and expressed as network requirements. After these initial steps, the best matching sub-systems are gathered into a full-size network structure that at least partially fits the application requirements. Finally, the application can be modified to better exploit the optical sub-systems (or more generally, the hardware), or vice-versa. Both may also be optimized jointly in a co-design process [12].

In order to speedup this iterative process, a better communication model between application developers and networking engineers is necessary. Generally, network architects express the capacities of their architectures in terms of bisectional bandwidth, and/or latency while application

developers tend to describe an application as bandwidth sensitive or latency sensitive. These generic definitions unfortunately provide limited information. In fact, two architectures providing the same theoretical bandwidth can result in different application performances. This is especially true in an optical networking context, as their buffer-less nature may temporarily impair the nominal bandwidth due to necessary retransmissions. Recent studies have shown that parallel computer architectures can be affected by long tail latency distributions [13]. However, such a statistic is difficult to obtain, as the latency is application dependent.

In this paper, we propose a new abstract model to address these issues. We start by representing applications as a set of interdependent computation and communication operations. Some computations can take place only after the reception of input data, and resulting communications arise only after the execution phase completes. This approach is similar to the skeleton-based model used by Jannsen et al [14]. However, while skeletons can explicitly represent an application code, they can be labor-intensive or unrealizable, especially for data-driven applications that exhibit irregular behavior. Unlike skeletons, which are derived from real applications, we create random operation sets. By changing the method of random set creation, we can change the type of traffic, the length of the computations, and their interdependencies. We can therefore mimic different classes of applications. In particular, by creating random operation sequences based on random tree structures, we intend to capture the various interdependencies between computations and communications present in irregular applications. The main goal of the proposed scheme is to capture the particular structure and dependencies of application traffic without involving the application itself. Moreover, given that the simulator is simplified at the traffic level, this releases a margin in terms of execution time and software complexity. This margin can in turn typically be exploited to include more optics specific details in the network simulation.

The remainder of this paper is organized as follows. In the next section we compare our approach with existing methodologies. In Section III, we present a model for the task generator of a HPC system and show how it can be parameterized. We apply generated operation sequences to several interconnection architectures in Section IV. In Section V we discuss how our approach can be extended. Conclusions are drawn in Section VI.

II. RELATED WORK

Due to the complexity, it is impossible to formulate the interconnect design as a solvable equation set. The design methodology consists of iteratively 1) sketching a preliminary design based on the estimated application requirements; 2) optimizing the interconnect parameters while keeping the design physically achievable; 3) evaluating the performance with modeling, simulation, and/or prototypes [15]. Analytical modeling is of limited use due to dynamic behavior such as congestion-aware or adaptive routing. Simulation is therefore generally considered the main evaluation technique, along with test-beds and prototypes later in the design cycle. However, in this paper we focus only on simulation.

Candidate networks can be simulated with simple traffic models, such as Poisson, which assumes no correlation between message arrivals. This gives good insight as to the intrinsic network performance in a dynamic environment (e.g. bandwidth, minimal latency). However, because most application traffic significantly diverges from Poisson traffic, it is difficult to obtain information as to how well the candidate network fits with any application. More complex packet or flow based generation methods have been proposed, in particular to capture the self-similar nature of the traffic in many networks [16]. Soteriou et al. proposed this type of traffic model for networks-on-chip [17]. Another packet generation technique consists of reading trace files of previously executed applications [18]. These complex traffic models, or trace based packet generators, produce flows with similar characteristics to those induced by applications. However, the traffic is replicated *a posteriori* - this method does not allow analysis of the impact of a message's delay on the application state or on future packet emissions.

To capture these dependencies, a network can be modeled within a cycle accurate simulation platform, such as gem5 [19][20]. On these platforms, HPC sub-systems are described at the instruction set level. Code execution, and therefore all application/network interdependences can be simulated. Because the interconnect is stressed in a realistic way, this cycle accurate approach provides good system scale performance estimates. However, the network's effect on performance might be difficult to isolate, as many elements of the system are modeled simultaneously. Furthermore, such extensive simulations are extremely complex and time consuming to construct and execute, and near impossible for extreme scale. For this reason, cycle accurate simulators tend to model the interconnect with a low level of detail (e.g. accounting only for a given delay when a message has to be transported, disregarding the network state).

To capture application/network interdependence while keeping the simulation more tractable, transformed computer code is executed along a coarse-grained network simulator [14]. The portion of code that is purely computation is replaced by a jump in time or a reduced model of the computation, while the network simulator handles communication calls. In this way, the communication patterns can be simulated without significant overhead, while remaining consistent with the original application.

Another simulation approach is to execute application derived benchmarks on real hardware in order to understand the requirements in terms of I/O [21]. This method provides exact measurements, both in number of bits and execution time, both extremely useful statistics, and can actually be used to obtain the aforementioned traffic traces. Similar to our proposed synthetic model, some benchmarks can be configured to stress different aspects of the interconnect. However, this approach is unfortunately only available to HPC systems users. Additionally, as the hardware is "concrete" - different protocols cannot be simulated in rapid succession (the entire system must be reconfigured), and hardware changes are simply inconceivable. Finally, *in situ* benchmarks can be used only in rare exceptions to monitor the activity of individual interconnect components, as probes are difficult to insert directly into the HPC system.

These performance evaluation approaches are generally complementary due to the different levels of software infrastructure they require. Packet generation models can be implemented in very lightweight software blocks, and therefore quickly adapted on top of any network simulator. This flexibility places them first in line to test novel interconnect concepts. Application simulators require a greater time investment: new interconnect architectures must be developed and integrated, simulations conducted, and relevant results extracted. These application simulators will typically be used as second-pass performance analysis. The output of a given performance evaluation approach can also be used as input for another approach. For example, as mentioned in the introduction, basic results in terms of bandwidth and latency can first be obtained through packet simulation. These bandwidth and latency values are then used as characteristics of the simplified interconnect employed in cycle accurate simulations. The opposite direction is also plausible - cycle accurate simulations can be used to obtain traces. Fig. 1 summarizes these approaches and their potential interactions.

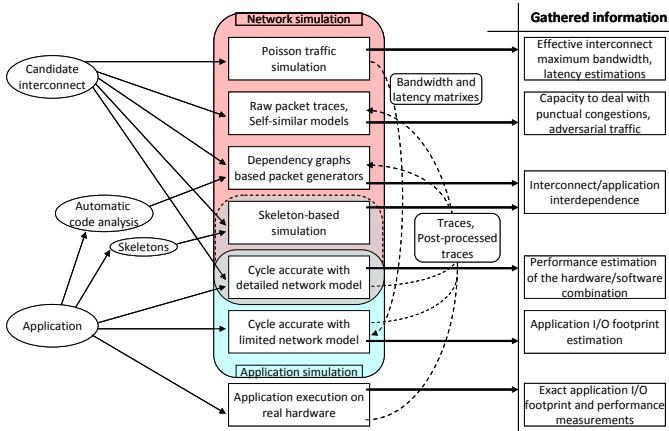


Fig. 1. Review of interconnect traffic simulation approaches.

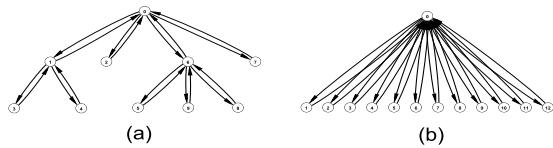


Fig. 2. Two examples of tree-represented tasks

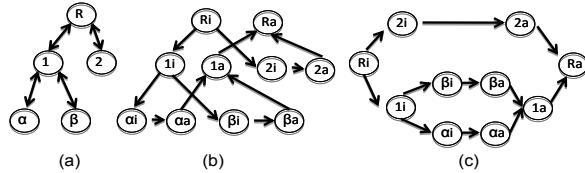


Fig. 3. Transformation of a tree task graph in a dependency graph

This description leaves a gap between lightweight independent packet based generators and accurate communication pattern simulators that capture the packet interdependencies but require more software development. A mechanism that both is easy to implement on top of network simulators and accurately generates correlated packets is desirable. For our purposes, this requirement exists in the context of optical networking. Optics require the simulator to

include physical layer details because designing optical networks requires coordination across different layers of the standard network protocol stack. Additionally, the buffer-less nature of optical networks greatly impacts transmission latencies. The optical network designer must have a tool to analyze these atypical features specific to optical networks. A related system was realized with the PhoenixSim NoC simulator. Dependency graphs representing computation and communication events and their dependencies are read by the simulator and packets generated accordingly [22]. Here we extend this concept and propose a random generation of inter-dependent events.

III. MODEL DESCRIPTION

A. Generation description

We consider an HPC system with N interconnected nodes which receives jobs at random points in time. We assume a node to be an independent CPU. This system is composed of $N-1$ computing nodes, and one *scheduler* node. When such a job is received, the *scheduler* assigns it to one node of the system (let this be called node 1). If no node is idle, the task is queued at the *scheduler*.

Upon job reception, node 1 executes its *root-task*, starting with the *initialization* phase. Each task may or may not contain independent subtasks. Once the *initialization* phase terminates, node 1 is aware of the number (and type) of subtasks to execute. If there is more than one subtask, node 1 contacts the *scheduler* and asks if idle nodes can be assigned to these subtasks to speedup the task completion time. If such idle nodes exist, the *scheduler* returns a list of idle nodes. Node 1 then contacts each node on this list and delegates a subtask i.e. sends the data required to start the execution of each subtask. A subtask can in turn be composed of more subtasks. The same scheme repeats recursively, eventually reaching *leaf* tasks which have no subtasks. Leaf tasks are also composed of an *initialization* phase, but this is instead directly followed by an *aggregation* phase. Once this aggregation phase terminates, the aggregated results are transmitted back to the delegating node, or are simply kept locally if the subtask is executed locally.

A task comprised of subtasks similarly has an *aggregation* phase that can be executed only when all results of the subtasks are available. If one or more subtasks have been delegated, the parent task might have to wait for the results. As soon as all of these results are obtained, this *aggregation* phase starts. Upon completion, results are transmitted to the delegating node if the completed task is a subtask. If the task is on the contrary the *root task*, we assume that no transmission takes place. In a more sophisticated version of the model, one node of the system could be designated as the I/O manager, to which final task results would be directed.

As it appears through the words (root, leaves), each randomly arising *job* has a tree structure, with a root-task and subtasks as vertices. The dependencies between the tasks form the tree branches. Downward facing edges (toward the leaves) indicate that subtasks cannot start before the parent task's initialization phase ends; upward facing edges (toward the root) imply that the task's aggregation phase depends on the completion of the subtasks. Fig. 2 provides graphical

examples of such trees. In particular, Fig. 2b represents a scatter/gather operation, extensively used in data-centers [7]. The tree task representation detailed here is not directly a dependency graph as evoked in [22]. However, such a dependency graph G' can be obtained by applying the following graph transformation to a job tree G . a) Each vertex, v in G , is divided into two vertices v_i and v_a , representing the initialization and aggregation phases respectively. b) v_i and v_a inherit v 's outgoing and incoming edges respectively. c) If v is a leaf, an additional edge $v_i \rightarrow v_a$ is added as the initialization phase precedes the aggregation phase. An example of such a transformation is depicted in Fig. 3.

B. Random task and trees generation

To generate the tree structure, we employ either a deterministic method or a stochastic one. The deterministic tree generator takes a vector, $z = [z_0, z_1, \dots, z_i]$, as a parameter, and constructs the tree such that each vertex of the i^{th} level of the tree has z_i siblings. Fig 4. displays several resulting trees for different z vectors. The stochastic method is inspired from the preferential attachment model described by Barabasi et al. [24]. Given a tree graph H , we obtain a new graph H' by adding a new vertex to H and by connecting it to exactly one vertex u already in H . To select u , we compute for each vertex v in H a weight $w(v) = (2\deg(v))^s$ where $\deg(v)$ designates the degree of v and s is a parameter. We then operate a weighted random selection. To obtain a tree with n vertices, we create an initial graph with only a single vertex and then apply our $H \rightarrow H'$ transformation $n-1$ times. Several graphs obtained with this method are displayed in Fig. 5. The parameter s is used to change the shape of the graph. When $s \geq 1$ graphs with few hub vertices are favored. In opposition, when $s < 1$ vertices with high degree are discriminated, leading to deeper trees with many levels.

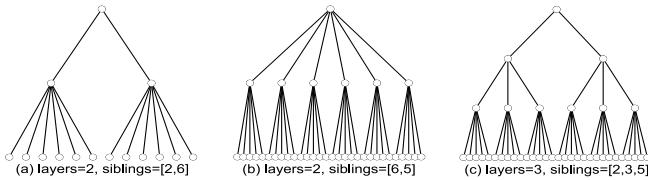


Fig. 4. Examples of trees provided by the deterministic tree generator

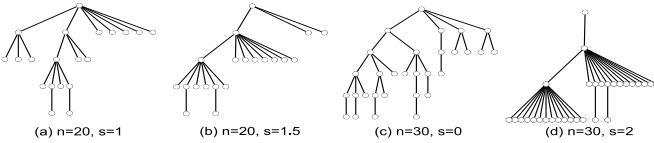


Fig. 5. Examples of trees delivered by the stochastic generator

Once the tree is generated, we create as many tasks as vertices in the tree. Formally, a task is described with four values $(\hat{s}_s, \hat{s}_r, \hat{t}_i, \hat{t}_a)$ where \hat{s}_s stands for the number of bits sent when the task is assigned (the data required to start the initialization phase), \hat{s}_r for the size in bits of the task results, \hat{t}_i and \hat{t}_a for the initialization and aggregation phase completion times (in seconds)². Random tasks (i.e. tree

²As we assume all compute nodes of the system to be homogenous, execution phases can be expressed in time units. In a heterogeneous environment, these would be expressed in terms of operations per second, to be later converted into time depending on each node's computing power.

vertices) can be obtained by sampling four *random variables* (S_s, S_r, T_i, T_a) . Our two *job* generators take parameters, (z, S_s, S_r, T_i, T_a) and $(n, s, S_s, S_r, T_i, T_a)$. Note that to obtain a wider diversity of jobs, (n, s) or z can also be randomly generated. Also note that special sizes are used for the root-task's messages. In the absence of an I/O node, these can be considered control messages.

C. Scheduler and delegation model

As previously introduced, the *scheduler* assigns tasks and subtasks. A possible *scheduler* implementation, realized in a similar way in [24], is detailed here. However, other scheduling strategies are possible.

Our example *scheduler* is the only entry point for arising *jobs*. It maintains a list of flags indicating whether a computing node is allocated or not. It also contains a FIFO for queued jobs. When a task begins, a computing node is randomly selected from those available, the description of the task is sent to this node, and the corresponding busy status is updated on the *scheduler*. If no node is available, the task is inserted into the FIFO. The *scheduler* then reacts to node allocation requests when necessary- as computing node M terminates the initialization phase of its assigned task, it may ask for additional nodes to complete the subtask execution. In our example, M asks for a number of nodes equal to the number of detected subtasks minus one, keeping one task for itself (in this way, node M keeps itself at least partially busy while expecting results from its delegates). The *scheduler*, upon reception of an allocation request for k nodes, allocates (randomly) k nodes if k nodes are currently available or all the remaining idling nodes else. The allocated nodes are marked as busy and their IDs sent to M . Reversely, as M detects that an allocated node is not needed anymore, it sends its ID back to the *scheduler*. If M was assigned to a root task, it replies to the *scheduler*, too. In both cases, the *scheduler* updates its status list and checks the FIFO. If a task is waiting, it is immediately assigned to the just released node.

Suppose node M is assigned to a task with m subtasks; node M will request $(m-1)$ allocations to the *scheduler*. If $k < m-1$ IDs for idle nodes come back, all subtasks cannot be executed in parallel. Node M therefore sends subtasks 1 through k to each of the k allocated nodes, then execute the $(k+1)^{\text{th}}$ subtask itself. Upon completion of this subtask, it assigns the $(k+2)^{\text{th}}$ subtask to itself but does not execute it until k answers are received from the k associated nodes. Each time an answer is received, say from node r , an unassigned subtask, if any remain, is sent back to r . If no there are no additional subtasks, a release message is sent to the *scheduler* for node r . Once all of the delegates of node M are either released or assigned to a new task, node M starts the execution of the $(k+2)^{\text{th}}$ task. This scheme is repeated until all subtasks are either delegated or executed locally.

D. Computation and communication footprint estimation

The computation and communication footprints of our model can be predicted (deterministic tree generator, constant distributions for S_s, S_r, T_i, T_a) or estimated. For the sake of notation simplicity, s_s, s_r, t_i and t_a hold for the expectancies of S_s, S_r, T_i, T_a respectively. \hat{n} is the expectancy of the number of vertices in the tree. If the deterministic generator is used,

$$\tilde{n} = 1 + \sum_{k=1}^{\text{length}(z)-1} \prod_{m=1}^k z_m$$

The computation time required by a single task is (t_i+t_a) . Therefore, the total computation footprint for a compound task is $F_x = \tilde{n}(t_i+t_a)$ (to distinguish between communication and computation footprints, x stands for computation (execution) while c stands for communication). Assuming β jobs arrivals per second on average, the mean computing load submitted to the HPC is $F_x\beta$. The relative computing load can be expressed as $\rho_x = F_x\beta/(N-1) = \tilde{n}(t_i+t_a)\beta/(N-1)$. ρ_x is the ratio between the offered computation load and the theoretical maximum computational capabilities. It is therefore also an upper bound for the computing nodes' utilization.

The communication footprint is more complicated to estimate. In case of system saturation, fewer nodes are allocated and some subtasks are executed sequentially and locally, which results in fewer communications between nodes. The communication footprint is also determined by the subtask allocation strategy. We consider here the strategy detailed in previous subsection and assume the situation where each node keeps always exactly one subtask locally ($k = m-1$ in previous subsection). The total number of communication phases between the delegating nodes and the nodes to which tasks are delegated is equal to the number of the leaves in the tree minus one. This fact is true for both stochastically and deterministically generated trees and is easily illustrated with an example. We will examine the case of a deterministic tree (Figure 4a) with z vector [2,6]. First, the root node keeps one subtask for itself but must communicate the other subtask to an idle node. Therefore to move from level 0 to 1 of the tree, there is one communication. To move from level 1 to 2, each of the two subtasks spawns six sub-subtasks. However each of the two nodes at level 1 keep one of their subtasks to execute locally. Therefore, moving from level 1 to 2 of the tree results in ten communications. Altogether, there are 11 communications, which is the number of leaves minus one. It should be noted that if a task spawns only one subtask, there is no communication because the parent task will execute this one subtask locally as previously explained. Therefore, each job induces a network traffic of $(L-1)(s_s+s_r)$, where L is the number of leaves in the tree. The number of exchanges with the *scheduler* also depends on the tree structure: only the vertices with degree > 2 induce a delegation process (except the root node that has no parent and delegates if the degree is > 1). Assuming d delegating vertices, $(L-1) + 2d + 2$ control messages are exchanged. The $L-1$ ones corresponds to the release messages sent after each subtask completes, the $2d$ accounts for the request-IDs and IDs-granted messages exchanged with the mapper; 2 more messages account for the communication with the root node to start and end the job. The total communication footprint for a task, under the maximum delegation assumption, is therefore $F_c = (L-1)(s_s+s_r+\epsilon) + \epsilon(2+2d)$, ϵ being the size of a control message. The expressions or methods to obtain L and d for each tree generator are provided in the Appendix.

Still considering the case where all potential delegations happen (there are enough idle nodes when requested), the average load offered to the interconnect is βF_c . Assuming each node has a maximum access bandwidth of B (bits/s), the

relative communication load can be expressed as $\rho_x = \beta F_c / BN$. Similarly to ρ_x , ρ_c provides a relation between the communication load induced by the arriving tasks and the interconnect capabilities. However, as tasks can be kept locally, ρ_c is a bound and not a strict prediction.

E. Rough bandwidth estimation for balanced system

In large systems the ratio between the communication and computation loads $\zeta = \rho_x/\rho_c = [\beta F_x/(N-1)]/[\beta F_c/BN]$ can be estimated as $\zeta \approx B/F$, where $F = F_c/F_x$. In the particular case where $\zeta = 1$, i.e. when the interconnect and the computing nodes are equally over- or under- dimensioned, we obtain $B = F$. In other words, F is the bandwidth required to have a bandwidth/computation balanced system. This definition of a balanced system considers *a utilization point of view only*. In fact, diverse application (or, in our case, generated job) requirements call for different bandwidth/computing resource ratios. Nevertheless, F is useful in expressing this ratio. If a bandwidth $B = \zeta F$ is present in the system, this means that the interconnect is ζ times more provisioned. As we will see in the next Section, varying ζ allows shorter job completion times at the price of network utilization, or vice-versa.

Table 1 summarizes the model's parameters and notation

F. Speedup bound

The completion time of a job arising in a HPC system depends on the properties of tasks associated with that job, and on the state of the system. F_x can be taken as a reference point, as it is the completion time if all tasks and subtasks are executed sequentially. Another reference point is the minimal completion time, which is given by the most costly path from the root vertex in the tree (considering t_i, t_a as vertex costs). For a leaf task to be executed, the initialization phase of each vertex separating it from the root must be completed. As each of these phases depends on the parent completion, they cannot be parallelized. Successive aggregation phases leading up to the root cannot be parallelized either. The minimal completion time can therefore be estimated with $D(t_i+t_a)$, where D is the depth of the tree. The maximal speedup compared to sequential execution can also be estimated as $F_x/D(t_i+t_a) = \tilde{n}/D$ (as $F_x = \tilde{n}(t_i+t_a)$).

TABLE I.

HPC system parameters	
N	Number of computing nodes in the system
B	Computing node global I/O bandwidth [bits/s]
ϵ	Control message size [bits]
Task generator parameters	
s_s	Average task assignment message size [bits]
s_r	Average task termination message size [bits]
t_i	Average task initialization computing time [s]
t_a	Average task aggregation computing time [s]
β	Average task arrival rate [s^{-1}]
\tilde{n}	Average number of subtasks in a job
L	Average number of leaves in a job tree
d	Average number of delegating tasks in a job
F_x	Per job average execution footprint
F_c	Per job average maximal communication footprint
F	Job generator footprint ratio - Equilibrated bandwidth reference [bits/s]
Deterministic tree generator parameter	
z	Vector of integers describing the tree span at each layer
Stochastic tree generator parameters	
n	Vertices in the tree

<i>s</i>	Tree shape parameter
Global bounds and metrics	
ρ_c	Maximum relative communication load - Maximal theoretical network utilization - Upper bound for measured network utilization
$\tilde{\rho}_c$	Measured node utilization
ρ_x	Relative computing load - theoretical node utilization - Upper bound for measured node utilization
$\tilde{\rho}_x$	Measured network utilization
ζ	System assymetry

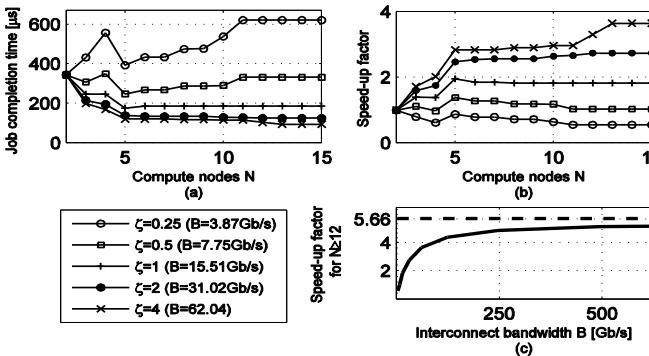


Fig. 6. (a) Time required to execute one $z=[4,3]$ task on different architectures with various B and N . (b) Corresponding speedup factors. (c) Evolution of the speedup with the bandwidth when $N=12$.

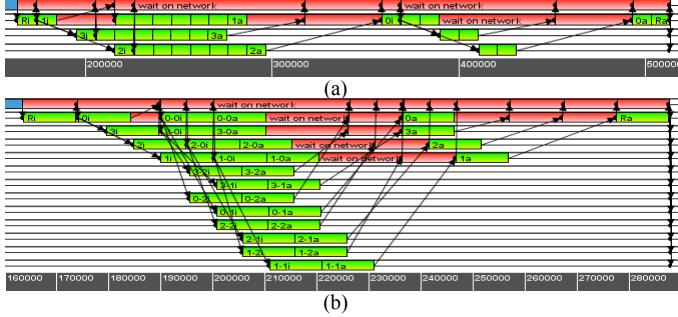


Fig. 7. (a) Simulation of one single task ($\text{det}([4,3])$) on $N=4$ nodes, (b) same task on a $N=13$

IV. SIMULATION EXAMPLES AND RESULTS

The *scheduler* and tree generators are implemented in the LWSim simulator [15]. In our implementation, the *scheduler* is also responsible for simulating the sporadic arrival of jobs. This is achieved by generating exponentially distributed inter-arrival times with parameter β . In the following reported simulations, 5 sequences of 1500 job arrivals have been simulated, using different pseudo-random number generator seeds.

A. Implementation validation and relations to the bounds

Simple configurations are first tested to validate our implementation. A simplified system organized around a single switch interconnecting the N computing nodes with bandwidth B is considered. The latency of the switch is fixed to 200ns, the latency of the links to 20ns (so about 4m assuming a light propagation speed of $2e8$ m/s). The predictability of the deterministic generator is exploited to validate the model: $z=[4,3]$ which translates into the following values $\tilde{n} = 17$, $L = 12$, $d = 5$. Constant values of 20kB (kBytes) and 40kB are used for s_s and s_t while $t_i = t_a = 10\mu\text{s}$. The size of control messages ε is assumed to be 40bytes. This

results in footprints per job of $F_x = 340\mu\text{s}$ and $F_c = 660.92kB$, with $F = 15.55$ Gb/s as equilibrated bandwidth value ($\zeta=1$).

The completion time of a single job running on networks with varying bandwidths is represented in Fig. 6a. It is measured as the time elapsed between the job arrival and the "job complete" message arrival at the *scheduler*. For $N=2$ (sequential execution on a single computing node) this time is $340\mu\text{s}$ plus a round trip time between the computing node and the scheduler, as expected. For higher N , completion times (and speedups, Fig. 6b) depend on how the $z=[4,3]$ job is *scheduled* over the N nodes. As the job tree has a depth of 3 and all tasks are equal, the maximum speedup is $\tilde{n}/D = 17/3$. For a given B , this speedup is maximized as soon as all possible delegations occur ($N \geq L$). The speedup increases to the maximum value as B increases (Fig. 6c). As it appears in Fig. 6a, some configurations are clearly disadvantageous and show the limit of the job allocation strategy, especially for low B . The case when $N=4$ is a typical example (Fig. 7a). Out of the first 4 subtasks, only three can be parallelized. Therefore, node 1 keeps the fourth task, and executes it after reception of the other three aggregations results. The three siblings of the last subtask are parallelized but because the network is slow, this parallelism leads to critical waiting times. Fig. 7b shows a case where subtask can be distributed more advantageously. Note that the dependency graph structure detailed in Fig. 3c is clearly visible in Fig. 7b.

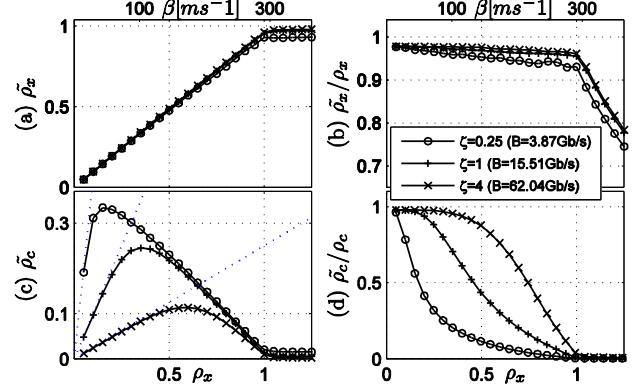


Fig. 8. Evolution of the measured utilizations: (a) computing node (c) network. (b) and (d) Comparisons with the theoretical values and bounds.

We then tested our implementation with $N=100$ and different task arrival times. We calculated values for the arrival rate using $\beta = \rho_x(N-1)/F_x$ such that $\rho_x = 10\%$, 20% , etc. For $\rho_x=0.1$, $\beta=29117$. We use network bandwidth proportional to the balanced bandwidth F to obtain $\rho_x = \rho_c$ ($\zeta=1$). Fig. 8 shows, for these increasing arrival rates (top axis) and loads (bottom axis) that: a) $\tilde{\rho}_x$ (i.e. the node utilization or the computation throughput) is increasing almost proportionally until $\rho_x \approx 1$, then saturates. The higher the bandwidth, the closer it approaches one. b) The deviation between the offered computing load ρ_x (which is also the predicted node utilization) and the effective node utilization $\tilde{\rho}_x$ is limited up to $\rho_x \approx 1$. Beyond this point, the deviation becomes obviously larger as the system saturates. c) The effective network utilization $\tilde{\rho}_c$ first grows linearly but drops much earlier as some computations are achieved sequentially (ρ_c is different for each different bandwidth- the corresponding trends are

visible is dotted lines). d) The effective network utilization deviates early from the bound except when bandwidth is over-provisioned ($\zeta=4$).

B. Utilization vs. speedup trade-off analysis

We tested our model with most of the parameters taken randomly and assuming a larger scale system. Task execution times and message sizes are modeled by uniform distributions [2 μ s, 100 μ s] and [2kB, 128kB] respectively. Random trees are generated with the stochastic generator. The number of tasks in the tree is in the interval [25,175] (so $\tilde{n}=100$) and s is fixed to one initially. ε is kept to 40bytes. F_x is therefore 10.2ms and F_c is 9.07 MB. These values give an equilibrated bandwidth $B=F=7.11$ Gb/s. We consider architectures with $N=50$ to 1000 computing nodes, and a job arrival rate β of 5000 task per second. We therefore have a computational load spanning between $\rho_x=1.04$ (for $N=50$) and $\rho_x=0.051$ ($N=1000$).

Figure 9(a) shows the $\tilde{\rho}_x/\rho_x$ deviation. The measured load is lower than expected for a system close to saturation (ρ_x close to 1) or equivalently with a slow network. Fig. 9(b) represents the speedup factor achieved by the system. For $B=10$ Gb/s and 100Gb/s, the speedup factor increases as more computing nodes are present in the system, until a point where idling nodes are always available and all possible delegations can take place. As more bandwidth generally decreases transmission times, node allocation and release messages suffer fewer delays, resulting in a more dynamic node allocation. Fewer nodes are hence required to reach the limit speedup. However, with $B=1$ Gb/s, the speedup is always less than one (counter-productive case) and is even decreasing as more nodes are added to the system. As visible in Fig. 9(c), this is not due to a network overload: the network utilization peaks around 23% for $N=200$ and then decreases. It is due to the wide intervals separating the node status update events and the actual start (or end) of the computation phase at the node. The relative time spent in this "synchronization overhead" depicted in Fig. 9(d). With $B=1$ Gb/s, allocated nodes are idling around 70% of the time, leading to poor performances. Note that high overheads are also measured with more bandwidth when the system is facing a more sustained load.

As it appears in the cases reported in Fig. 9, bandwidth (or, more specifically, the bandwidth over-provisioning factor ζ) is the determinant speed-up factor. This fact is confirmed in Fig. 10(a), which represents the speed-up/network utilization trade-off for the previous simulation cases. We can see that using the current scheduler, significant speed-up can only be achieved at the price of very low network utilization. In Figure 10(b) we compare two HPC systems ($N=300$ or 500) with different bandwidths (from 10Gb/s to 100Gb/s by steps of 10Gb/s) running two types of random tasks (s parameter 1 or 2). The speed-up/network utilization trade-off appears clearly again with this example. This parameter can therefore be used to mimic applications parallelizable to various extents.

C. Random tree based traffic over optical network

These presented results aim at explaining the raw characteristics of our random task traffic model. We now achieve our final objective: apply this traffic to different optical networking platforms, analyze how these platforms

cope with this new environment, and take appropriate measures if poor performances are reported.

We consider the same random tree task generator configuration, applied to a HPC system comprised of 64 nodes. These nodes are either connected with the single switch network structure (the reference case), or with a multistage optical interconnection network controlled by an in-band signaling mechanism called SPINet [8]. B is fixed to 40Gb/s in the reference case and to 160Gb/s in the SPINet case. Three different SPINet variants are tested: with the minimal number of interconnection stages (i.e. 6 for 64 ports) or with 1 or 2 additional stages. At the network level, the SPINet architecture provides a drastically lower average latency than the reference case (Fig. 11 (a-b)). The additional stages aid to further reduce this latency. However, at the application level, the benefits of the SPINet architecture, while still evident, are not of the same order. Moreover, the modest advantages in terms of latency gained by using multiple stages are not translated into a significant speedup. This demonstrates that optimizing a network considering only latency can be useless, or even counter-productive.

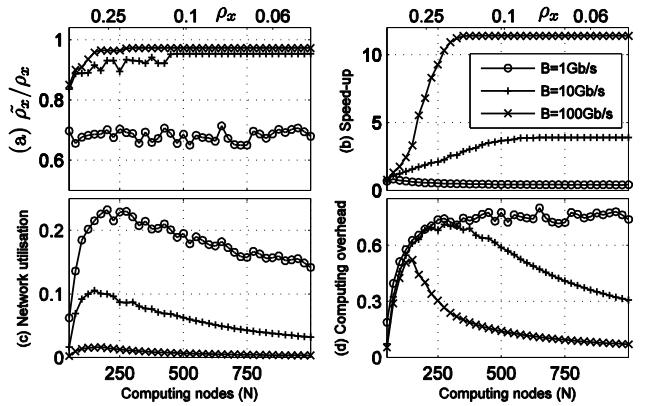


Fig. 9. Various measurement with $\beta=5$ task per ms and random tasks.

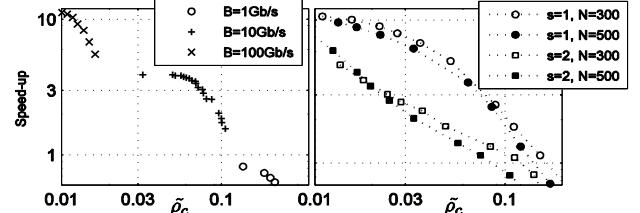


Fig. 10. Network-utilisation / speedup trade-offs.

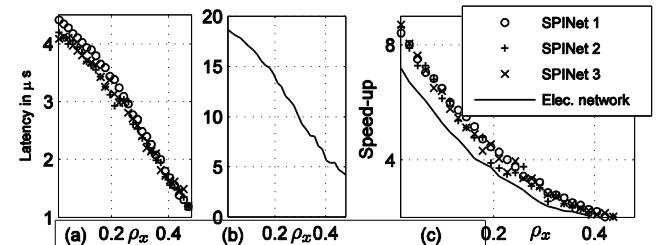


Fig. 11. Optical vs. electrical network comparison

V. EXTENSIONS AND FUTURE WORK

The proposed application model has been primarily developed to analyze the impact of unconventional optics-

aware novel protocols on interdependent communication phases. However this model also has large potential for extensions beyond those proposed in this work. The development of more complex, possibly optics-aware, scheduling schemes is one possible direction. Another extension could consist of replacing some vertices in the random tree with a well known dependency pattern (e.g. an FFT computation) to closer capture real application behaviors. Alternatively, a dependency graph could be generated not out of trees, but as directed graphs, leveraging an example such as the GGen generator detailed in reference [25]. The HPC model can also be complemented to consider multi-core nodes or I/O nodes.

Looking further ahead, such an approach can be leveraged to analyze the impacts of dynamic load-balancing or contention resolution scheme on the global performances. An interesting feature that emerges from the reported results is that the inter-dependence between the network and the scheduler is as important as the application/network relation. Scheduling algorithms design should therefore be included in the co-design approach.

VI. CONCLUSIONS

This paper presents a performance evaluation approach for High Performance Computing optical interconnects. Our approach allows us to capture the interdependencies between the application and network mechanisms. It also provides the capability to tune the injected network load. Our approach, being additionally relatively easy to implement on top of a network simulator, allows us to include more optics-specific aspects in the simulation while still evaluating the HPC system as a whole. A formal model describing our approach is provided. Numerical results obtained by simulation in various cases exemplify the situations where our approach can provide valuable information. In particular, results show that improvements in terms of raw latency do not always translate in global performance improvements.

APPENDIX

The number of leaves L and nodes with degree > 2 d are easy to express when using the deterministic tree generator.

$$L_{\text{det}} = \prod_{k=1}^{\text{length}(z)} z_k \quad d_{\text{det}} = 1 + \sum_{k=0}^{\text{length}(z)-2} \begin{cases} z_k & \text{if } z_k > 1 \\ 0 & \text{else} \end{cases}$$

i.e. L_{det} (deterministic tree generator) is the product of z elements (so 12, 30 and 30 for the examples of Fig. 4) and d_{det} the sum of the $n-1$ first elements of z that are greater than 1, plus one (so 3, 7, and 9 in Fig. 4). L_{sto} (stochastic generator) and d_{sto} are harder to derive. L_{sto} can be written as np_1 , and $d_{\text{sto}} = p_2 + (n-1)(1-p_1-p_2)$, where p_1 and p_2 denote the probability of a vertex in the tree to have degree 1 or 2 respectively, and $(1-p_1-p_2)$ is the probability of having a degree > 2 . A table of values for p_1 and p_2 can be obtained via Monte Carlo simulations for several values of n and s .

ACKNOWLEDGEMENT

Sandia National Laboratories is a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Company, for the United States Department of Energy's National Nuclear

Security Administration under contract DE-AC04-94AL85000.

REFERENCES

- [1] S. Borkar, "How to stop interconnects from hindering the future of computing", Optical Interconnects Conference, May 2013.
- [2] S. Scott, D. Abts, J.Kem and W.J. Dally "The BlackWidow High-Radix Clos Network," *Computer Architecture, ISCA '06*.
- [3] D. Abts, M. Marty, P. Wells, P. Klausler, H. Liu "Energy Proportional Datacenter Networks", *ISCA 2010*.
- [4] G. Hendry, "The Role of Optical Links in HPC System Interconnects", Optical Interconnects Conference, May 2013.
- [5] M. A. Taubenblatt, "Optical Interconnects for High-Performance Computing", *Journal of Lightwave Technology*, Vol. 3, no. 4, 2012.
- [6] N. Farrington, et al, "Helios: A hybrid Electrical/Optical Switch Architecture for Modular Data Centers", *ACM SIGCOMM 2010*.
- [7] G. Wang et al, "c-Through: part-time optics in data centers", *Proceedings of the ACM SIGCOMM 2010 conference*
- [8] G. Dongaonkar, et.al, "Ultra-low Latency Optical Switching for Short Message Sizes in Cluster Scale Systems", Optical Interconnects, 2013.
- [9] H. Wang, Y. Xia, K. Bergman, T.S.E. Ng, S. Sahu, K. Sripanidkulchai, "Rethinking the Physical Layer of Data Center Networks of the Next Decade: Using Optics to Enable Efficient *-Cast Connectivity," *ACM SIGCOMM Computer Communication Review* 43 (3) (Jul 2013).
- [10] S. Rumley, et al. " Low Latency, Rack Scale Optical Interconnection Networks for Rack Scale Data Center Applications", accepted for publication, ECOC 2013.
- [11] Y. Xiaohui, et al. "Fast barrier synchronization with AWGR-based optical switch in high-performance and parallel computing," *Optical Fiber Communication Conference and Exposition (OFC/NFOEC)*, 2011.
- [12] C. Chan, et al. "Software Design Space Exploration for Exascale Combustion Co-design*", LNCS 7905 , Springer, Berlin, 2013.
- [13] J. Dean, L. A. Barroso, "The tail at Scale"; *Communications of the ACM*, Volume 56 Issue 2, February 2013.
- [14] C. L. Janssen et al. "A simulator for large-scale parallel computer architectures", *IJDST*, vol.1 no. 2, pp. 57-73, 2010.
- [15] M. Glick, et al. " Modeling and Simulation Environment for Photonic Interconnection Networks in High Performance Computing", *IEEE ITCN 2013*.
- [16] V. Paxson and S. Floyd. "Wide-area traffic: the failure of Poisson modeling". *IEEE/ACM Trans. on Networking*, Vol. 3, No. 3, 1995.
- [17] V. Soteriou, H. Wang and L.-S. Peh, "A Statistical Traffic Model for On-Chip Interconnection Networks," *MASCOTS 2006*.
- [18] M. M. Tikir, et al. PSINS: An Open Source Event Tracer and Execution Simulator for MPI Applications. *15th Int. Euro Par Conf. On Parallel Proc.*, pages 135--148, 2009.
- [19] N. Binkert, et al. "The gem5 simulator", *SIGARCH Comput.Archit. News*, 2011.
- [20] A. Van Laer, et .al, "Full System Simulation of Optically Interconnected Chip Multiprocessors Using gem5", *OFC 2013*.
- [21] J. Borrill, L. Oliker, J. Shalf, H. Shan, "Investigation of Leading HPC I/O Performance Using a Scientific-Application Derived Benchmark", *SC07*.
- [22] G. Hendry, et al. "Circuit-Switched Memory Access in Photonic Interconnection Networks for High-Performance Embedded Computing", *Supercomputing (SC)*. (Nov 2010).
- [23] A.-L. Barabási, R. Albert "Emergence of Scaling in Random Networks" *Science 15: Vol. 286 no. 5439 pp. 509-512, October 1999*.
- [24] D. Zydek, G. Chmaj, S. Chiu, " Modeling computational limitations in H-Phy and Overlay-NoC architectures", *Springer Journal of Supercomputing*, April 2013.
- [25] D. Cordeiro, et al.. "Random graph generation for scheduling simulations". *SIMUTools '10*

