Eurographics Symposium on Parallel Graphics and Visualization (2012)
H. Childs, T. Kuhlen, and F. Marton (Editors)

# Shift-Based Parallel Image Compositing on InfiniBand™ Fat-Trees

Xavier Cavin and Olivier Demengeon

Inria, France

**Abstract**

*Parallel image compositing has been widely studied over the past 20 years, as this is one, if not the most, crucial element in the implementation of a scalable parallel rendering system. Many algorithms have been proposed and implemented on a large variety of supercomputers. Among the existing supercomputers, InfiniBand™ (IB) PC clusters, and their associated fat-tree topology, are clearly becoming the dominant architecture, as they provide the scalability, high bandwidth and low latency required by the most demanding parallel applications. Surprisingly, very few efforts have been devoted to the implementation and performance evaluation of parallel image compositing algorithms on this kind of architecture. We propose in this paper a new parallel image compositing algorithm, called Shift-Based, relying on a well-known communication pattern called shift permutation. Indeed, shift permutation is one of the possible ways to get the maximum cross bisectional bandwidth provided by an IB fat-tree cluster. We show that our Shift-Based algorithm scales on any number of processing nodes (with peak performance on specific counts), allows overlapping communications with computations and exhibits contention-free network communications. This is demonstrated with the image compositing of very high resolution images at interactive frame rates.*

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture—Parallel processing I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics I.3.8 [Computer Graphics]: Applications—Image compositing

## 1. Introduction

### 1.1. Parallel Rendering

Both in the industry and in the academic worlds, larger and larger numerical datasets are being produced on a daily basis, whether they are large Computer Aided Design (CAD) assemblies, results of numerical simulations or 3D scanned information, just to cite a few. Unless a considerable preprocessing can be applied to the data, a single computer cannot accomplish real-time rendering of these data. In most cases, high-end parallel visualization solutions must be designed to tackle these ever-increasing amounts of data, sometimes on-the-fly as they are being produced.

Sort-last parallel rendering, as introduced by Molnar et al. [MCEF94], is an efficient technique to render very large datasets in parallel. In this kind of parallel applications, the dataset to be visualized is first partitioned and distributed across multiple processors. Each processor renders its subset of the dataset independently using its local resources; this stage does not imply any communication between the processors and scales perfectly. Then the computed images are composed together by a parallel image compositing algorithm and the final image is delivered. The parallel image compositing stage requires a lot of inter-processor communication, and may become the bottleneck of a sort-last parallel pipeline if it is not designed with sufficient care.

Since the early 1990's, parallel image compositing has been widely studied, and numerous algorithms have been developed and experimented on many parallel computing architectures, ranging from supercomputers to PC clusters. These image compositing algorithms are commonly classified into three categories: Direct Send [Neu94], Binary Swap [MPHK94] and Parallel Pipeline [LRN96]. The two latest advances in the field include 2-3 Swap by Yu et al. [YWM08], which is an extension of the Binary Swap to non-power of two numbers of processors, and the Radix-k algorithm by Peterka et al. [PGR*09, MKPH11] which uni-



DOI: 10.2312/EGPGV/EGPGV12/129-138

fies the Direct Send and Binary Swap and enables numerous other configurations.

The 2-3 Swap and Radix-k algorithms have been successfully deployed on very large supercomputer configurations. Yu et al. report compositing times of 2 seconds per image for a 16 Mega-pixels (16 bytes per pixel) resolution with 1024 processors of a Cray XT4 system connected to a SeaStar 2 router. Peterka et.al report compositing times of 0.7 seconds per image for an 8 Mega-pixels (16 bytes per pixel) resolution with up to 34,816 processes on an IBM Blue Gene/P.

### 1.2. Multistage Interconnection Networks

A large majority of high performance computing clusters today rely on the central switch network architecture. This network architecture considerably simplifies the work of the programmer, since each node of the cluster can communicate with every other node. In the particular case where the central switch is a true crossbar, the network has almost ideal properties: constant latency between all pairs of nodes and full bisection bandwidth (any half of the nodes can simultaneously communicate with the other half at full speed).

Most parallel applications and communication libraries typically treat central switch network architectures as if they were true crossbar switches, and do not take into account the actual internal characteristics of the switch. However, practical central switches are generally implemented as multistage interconnection network (MIN), and do not truly provide the latency and full bisection bandwidth characteristics of crossbars, as shown in [HSL08]. In particular, the routing policies, needed to route connections in a MIN, can greatly affect the performance of a given communication pattern. The fat-tree topology is a class of MIN and is one of the most commonly used topologies for high performance computing clusters. In particular for InfiniBand™ (IB) based clusters, this topology is the dominating one. This includes large installations such as CEA Tera-100, LANL Roadrunner, TACC Ranger or FZJ-JuRoPA (Source: http://www.top500.org/).

For IB fat-trees, as with most other topologies, the routing algorithm is crucial for efficient use of the underlying topology. IB provides a deterministic static-destination routing that is computed offline and loaded into the different switches of the MIN. Several options are offered to the system administrator to compute the routing tables, depending on the physical network architecture of the cluster. A given communication pattern can either achieve satisfactory performance or on the contrary can lead to heavy contention for different routing strategies.

### 1.3. Our Contributions

In this paper, we start in Section 2 with an overview of fat-tree IB clusters, which represent a significant part of today's high-performance supercomputers, and with a presentation of existing parallel image compositing algorithms. Surprisingly, we have not found any study of these algorithms on this ever-growing class of supercomputers. In Section 3, we introduce a new parallel image compositing algorithm called Shift-Based, relying on a very well-known parallel communication pattern called shift permutation. Indeed, a lot of efforts have been dedicated to efficiently support this communication pattern on fat-tree IB clusters. We continue in Section 4 with an analysis of the theoretical communication and computation costs of our new algorithm, and we compare them to those of the existing parallel image compositing algorithms, in an ideal environment. Then, we show how network contention, appearing in real-life environments, can negatively affect the expected performance of these algorithms. In Section 5, we experiment our algorithm on a 144-node fat-tree IB cluster and we show how it maximizes the utilization of the bandwidth made available by the IB cluster. We demonstrate that our algorithm scales very well on any number of processors (with peak performance on specific counts) and performs at least twice better as compared to existing algorithms. We conclude and present future work in Section 6, anticipating similar performance results at larger scales.

## 2. Background

### 2.1. Fat-Tree IB Clusters

The dominant architecture of scalable high-performance supercomputers has clearly become the clustered architecture; the latest (36th) edition of the TOP500 list reports 414 clustered systems (82.8 %). For these clusters, as well as for other systems, a scalable, high bandwidth and low latency network is a key. The InfiniBand™ (IB) architecture standard [IBT04] meets all these requirements and is the interconnection network of choice for such systems: the same TOP500 list reports 213 IB based systems (42.6 %) in total, including 178 IB clusters.

Large IB clusters are commonly built with multiple fixed-arity crossbar switches (up to 36x 40 Gbps ports) and host-channel adapters (HCAs) providing up to 40 Gbps links. The fat-tree topology is the most commonly used to interconnect the nodes and IB switches, providing varying ratios of cross-bisectional bandwidth (CBB). Closely related to the chosen topology, the routing algorithm, used to route the data between switches from one node to the other, is of particular importance when one wants to maximize the utilization of the network.

We focus in this paper on large scale fat-tree IB clusters, as they are sufficiently representative of today's supercomputers, and we will review their main characteristics in the remaining of this Section. However, a similar analysis could be conducted on the other types of interconnection networks and supercomputers architectures.

#### 2.1.1. IB Clusters

The basic building blocks of large IB clusters are integrated non-blocking switch elements (i.e. crossbars) with a relatively low number of ports. As an example, QDR (40 Gbps) IB switches come in 8, 18 and 36 ports configurations. Crossbar switches exhibit almost ideal network properties: constant latency between all pairs of endpoints and full bisection bandwidth (any half of the nodes can simultaneously communicate with the other half at full speed, i.e. without contention).

The IB standard does not impose any particular network topology. In other words, switches can be connected together in any arbitrary way. IB switches use oblivious destination based distributed routing, meaning that a switch routes a packet out a particular port based solely on the destination address of the packet. This decision is determined via a simple static routing table (Random or Linear Forwarding Table) that defines which port leads to which endpoint. On startup, or after a topology change, a central entity called the Subnet Manager (SM) discovers the network topology using special packets computes the routing table for each switch and uploads them into the switches. As this operation requires network downtime, it cannot be performed too frequently.

The current implementation of OpenSM offers five routing engines: Min Hop, UPDN Unicast, Fat-tree Unicast [ZJKL10], LASH Unicast, DOR Unicast. Additionally, OpenSM also supports a file method which can load routes from a table. The interested reader can refer to the OpenSM documentation for more details about these methods.

#### 2.1.2. Fat-Tree Topology

The fat-tree topology is an indirect interconnection network based on a complete binary tree [Lei85]. The processors of a fat-tree are located at the leaves of a single-rooted complete binary tree, and the internal nodes are crossbar switches. Unlike traditional trees in computer science, fat-trees are more like real trees, because they get thicker near the root. Going up the fat-tree, the number of wires connecting a node with its father increases. The rate of growth influences the size and cost of the hardware, but also the communication bandwidth it can support. As the arity of the internal switches of the fat-tree increases as we get closer to the root, the physical implementation of this topology is impractical.

To overcome this limitation, the concept of a multi-rooted topology was introduced. The non-blocking nature of the single rooted tree is replaced by a weaker attribute: given a permutation of source and destination pairs, the routing on the tree can be made non-blocking (i.e. rearrangeably non-blocking). Several families of fat-trees have been introduced. The $k$-ary $n$-tree [PV97] is the basic type of tree built out of switches with an equal number of connections going up or down the tree. A $k$-ary $n$-tree is composed of $k^n$ processing nodes and $nk^{n-1}$ $k \times k$ constant arity commutation switches.

The $k$-ary $n$-tree topology has been extended with the introduction of Generalized Fat-Trees (GTF) which allow for a different number of up and down connections [OIDK95]. Extended Generalized Fat-Trees (XGFT) further extends the possible topologies allowing for different number of connections at each level. XGFTs cannot capture existing real life topologies, since they allow only a single connection between switches: the resulting family does not preserve cross bisectional bandwidth (the bandwidth towards the top of the tree is not equal to that of the leaves).

The latest topology introduced to represent fat-trees is the Parallel ports Generalized Fat Tree (PGFT) and its sub-classification into Real Life Fat-Tree (RLFT) [Zah10]. PGFT allows multiple connections between the switches and support a large variety of topologies, including some unpractical to build. RLFT is a subclass of PGFT with several restrictions (detailed in [Zah10]), and it encompasses the majority of real life configurations.

#### 2.1.3. Hot Spots in Fat-Tree IB Clusters

Fat-tree IB clusters with full cross-bisectional bandwidth have been measured to provide an effective bisection bandwidth in the range of 55-60 % for MPI communication patterns [HSL08]. This performance degradation is due to cases where multiple sources congest a particular network link, creating a hot spot. Indeed, even if enough physical links are available to support any communication pattern, the routing mechanism might oversubscribe some physical links while others remain idle. The probability of interconnect hot spots increases with the size of cluster and might degrade the latency and effective bandwidth experienced by MPI operations.

One possible solution to the hot-spot problem is the IB Lid Mask Control (LMC) mechanism. The LMC mechanism assigns multiple Local Identifiers (LIDs) to hosts and enables multiple routes for every pair of peers. However, for multiple reasons discussed in [HSL08], LMC-based routing strategies can be seen as counter-productive at large scale.

In this paper, we will focus on single-path static routing. Consequently, the chosen routing engine must be taken into account when designing a parallel algorithm in order to avoid hot spots and benefit from most of the theoretical cross-bisectional bandwidth.

#### 2.1.4. Non-blocking Shift Permutations

Given a set of $p$ nodes, shift permutation is defined as the set of source/destination pairs such that processing node $P_i$ sends data to processing node $P_j$:

$$\{(P_i, P_j) | j = (i+S) mod p\}_{i=0}^{i=p-1} \quad (1)$$

for $S \in \{1 \dots p-1\}$. Shift-based communication involves $p-1$ communication steps: at each step $S$, processing node $P_i$ sends data to processing node $P_{(i+S) mod p}$ and receives data from processing node $P_{(i-S+p) mod p}$.

Shift-based communication enables all nodes to send and receive data simultaneously in all steps and ensures a regular and predictive communication pattern. This particular pattern has been shown to be representative for scientific applications [KLJ08] and has been widely studied, on multiple network topologies including IB fat-trees.

In particular, the Fat-tree Unicast routing engine available in the latest version of OpenSM provides routing tables for contention free shift communication pattern on var-ary n-trees [ZJKL10], a variant of $k$-ary $n$-trees. For more complex, real life fat-trees (i.e. RLFT), $d$-mod-$k$ routing has recently been proven to provide non-blocking traffic for the same shift-permutations [Zah10].

In this paper, we will focus on var-ary n-trees IB clusters with Fat-tree Unicast routing enabled. However, we can anticipate similar results with $d$-mod-$k$ routing on RLFT. Our goal will be to design a parallel image compositing algorithm based on the shift permutation pattern, in order to benefit from the full available cross-bisectional bandwidth.

### 2.2. Parallel Image Compositing Algorithms

In this Section, we will review the existing parallel image compositing algorithms. We will assume $p$ processing nodes. Each processing node $P_i$ owns a vector $x_i$ (the local image) of $n$ components (the pixels). The goal of a parallel image compositing algorithm is to blend all $x_i$ with all $x_j$ using a component wise linear binary operator.

#### 2.2.1. Direct Send

The Direct Send algorithm [Neu94] is the most straightforward parallel image compositing algorithm to implement. Each node $P_i$ is responsible of compositing the $(n/p)$ pixels ranging from $i(n/p)$ to $(i+1)(n/p)-1$. The Direct Send algorithm requires a single stage of communication: during this stage, in the worst case, each node $P_i$ sends $(p-1)$ messages of $(n/p)$ pixels, one to each other node, and receives $(p-1)$ messages of $(n/p)$ pixels, one from each other node.

The main advantage of the Direct Send is its flexibility, since it can accommodate any number of processing nodes. It is very easy to implement and allows overlapping computations (i.e. pixels blending) with communications. However, it involves multiple processors sending messages to the same processor at the same time in an unpredictable and non-deterministic communication pattern.

#### 2.2.2. Binary Swap and 2-3 Swap

The Binary Swap algorithm [MPHK94] is based on a binary tree compositing strategy which keeps every node busy in all stages of the process. It takes $\log_2(p)$ stages to complete, where $p$ must be a power of two (i.e. $p=2^k$) to fully exploit parallelism.

At each stage $i$ $(0 < i < \log_2(p)+1)$ of the algorithm, each node sends one message of $n/2^i$ pixels to a pair node, and receives one message of $n/2^i$ pixels from the same pair node.

The Binary Swap algorithm exhibits a regular and predictable communication pattern. However, it does only support powers of two numbers of processors and its synchronous nature does not permit to overlap computations with communications (except on a Gigabit Ethernet network with a strongly streamed communication approach [CMF05]).

The 2-3 Swap algorithm [YWM08] extends the Binary Swap algorithm to non-powers of two numbers of processors, at the cost of increased latency and bandwidth requirements.

#### 2.2.3. Radix-k

The Radix-k algorithm [PGR*09] is a configurable image compositing algorithm which unifies the Binary Swap and Direct Send algorithms, and enables numerous other algorithms through appropriate choice of radices or $k$-values.

A Radix-k algorithm is completely specified by a vector $k=[k_1,k_2,\ldots,k_r]$ of $r$ values such that $p=\prod_{i=1}^{i=r} k_i$. It consists in $r$ communication/compositing rounds. During each round $i$, the $p$ processing nodes are divided into $p/k_i$ groups of $k_i$ participants, which communicate only within a group in a Direct Send fashion. When all of the $k$-values are equal to 2, this gives the Binary Swap algorithm. When $k=[p]$, there is only one single round and this gives the Direct Send algorithm.

The Radix-k algorithm combines the advantages and the drawbacks of the Direct Send and the Binary Swap algorithms. Indeed:

- It is not limited to powers of two numbers of processors.
- It allows to partially overlapping computations with communication within a round, but not between the rounds.
- The communication pattern is irregular and unpredictable within a given round.

It has been shown to perform better than the Binary Swap and 2-3 Swap algorithms.

#### 2.2.4. Parallel Pipeline

The Parallel Pipeline algorithm [LRN96] requires $(p-1)$ communication/computation rounds. The processing nodes are organized on a circular ring. At each round, each processing node composes the $(n/p)$ pixels it has received from its "previous" neighbor in the ring and sends these $(n/p)$ composed pixels to its "next" neighbor in the ring. The subimages are accumulated in a pipelined fashion along the ring.

The Parallel Pipeline algorithm presents two main advantages. First, it can support any number of processors. Second, it can benefit from the full-bisection bandwidth provided by a fat-tree IB cluster, since each node always sends

data to the same node: contention free communication can be achieved using Fat-tree Unicast routing if the processes are placed with sufficient care.

However, similarly to the Binary Swap algorithm, its synchronous nature does not permit to overlap computations with communications.

## 3. Shift-Based Image Compositing

We introduce in this Section our new parallel image compositing algorithm, called Shift-Based. We rely on the analysis done in the previous Section to build a new algorithm that combines all the advantages of existing image compositing methods. In particular, we base it on a shift-permutation pattern to ensure full utilization of the cross-bisectional bandwidth made available by fat-tree IB clusters.

### 3.1. Algorithm Requirements

Our algorithm is based on the five following requirements:

1. It should keep all processors busy and load balanced at all stages.
2. It should support any number of processors, such as the Direct Send, the 2-3 Swap and the Radix-k algorithms.
3. Computations and communications should be overlapped, such as the Direct Send and partially the Radix-k algorithms.
4. It should provide contention free communication on IB fat-trees, such as the Parallel Pipeline algorithm.
5. Finally, it must be easy to understand and simple to implement.

### 3.2. Algorithm Description

We make no assumption on the number $p$ of nodes, which can be any value greater than 1 (requirement 2). In order to fulfill requirement 4, we base our algorithm on a shift permutation pattern. Similarly to the Direct Send method, each processing node $P_i$ is responsible of compositing the $(n/p)$ pixels ranging from $i(n/p)$ to $(i+1)(n/p)-1$. Our algorithm simply consists in $(p-1)$ compositing stages, as illustrated by Figure 1. During each stage $s$ $(0 < s < p)$, each processing node $P_i$:

- Sends $(n/p)$ pixels to processing node $P_{(i+s)modp}$ and receives $(n/p)$ pixels from processing node $P_{(i-s+p)modp}$.
- Composes the pixels received during previous stage $s-1$ (for $s > 1$) while it does the send and receive operations.

It is easy to understand that this simple algorithm (requirement 5) keeps all processors busy and load balanced at all stages (requirement 1) and allows to overlap computations with communications (requirement 3).

Similarly to the Direct Send and Radix-k algorithms, partial subimages can arrive in a non-continuous order (see the example of processing node $P_0$ in Figure 1). In these cases, the non-continuous subimage is simply buffered until the missing subimage has been received (see for example at the end of Stage 3 for processing node $P_0$ in Figure 1).

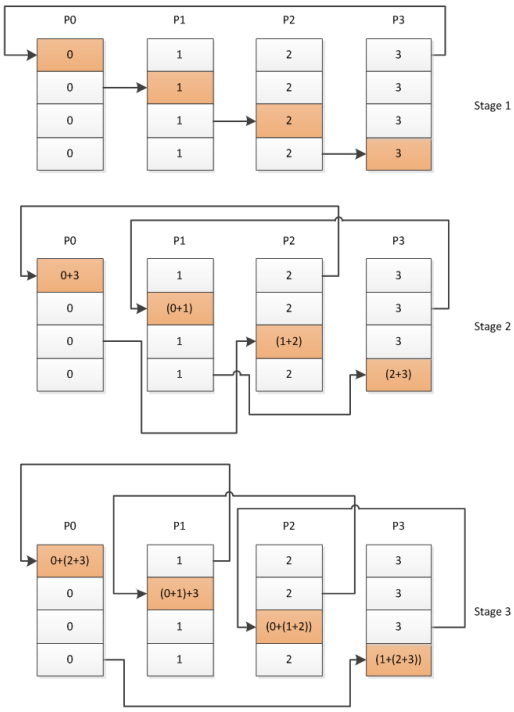


**Figure 1:** *The three stages of our Shift-Based parallel image compositing algorithm with $p = 4$ processing nodes.*

## 4. Communication and Computation costs

Similarly to Peterka et al. [PGR*09], we assume a simple cost model as proposed by Chan et al. [CHPvdG07] to analyze the communication and computation costs of the different existing parallel image compositing algorithms. We recall here the main assumptions:

- The target architecture is a distributed memory parallel architecture composed of $p$ computational nodes indexed from $P_0$ to $P_{p-1}$.
- The interconnection network is fully connected: each node can send directly to any other node.
- At any given time, a single node can send only one message to one other node. Similarly, it can only receive one message from one other node. It is also assumed that a node can send and receive simultaneously.
- The communication cost of sending a message between two nodes is modeled by $\alpha + n\beta$ in the absence of network conflict. Here, $\alpha$ represents the message startup time (i.e. latency) and $\beta$ represents the per data item transmission

time (i.e. reciprocal of bandwidth). The cost of the message is not a function of the distance between nodes.
- Messages traveling in opposite directions on a link do not conflict.
- The computation time to reduce one data item is denoted by $\gamma$.

We consider the worst case scenario, where all pixels are active, and we do not consider optimizations to take advantage of the sparseness of the partial images.

The latency term of the Direct Send is bounded by $\alpha(p-1)$, since each node has to send at most $(p-1)$ messages. The communication term is bounded by $\beta(p-1)(n/p)$, since each message contains $(n/p)$ pixels.

The latency term of the Binary Swap is $\alpha\log_2(p)$, since there are $log_2(p)$ stages of communication. The communication term is equal to: $\beta\sum_{i=1}^{i=\log_2(p)} n/2^i = \beta n(p-1)/p$.

The upper bounds of the latency and communication terms of the 2-3 Swap are the ones reported in [YWM08], and correspond to the non-power of two case.

The communication term of the Radix-k is the one reported in [PGR*09], and can be easily verified. On the contrary, we disagree with the value given by the authors for the latency term. Indeed, during each round, there are $k_i - 1$ messages exchanged, and they should be summed up for the r rounds. We claim that the latency term is then: $\alpha\sum_{i=1}^{i=r}(k_i-1)$. It can be easily verified that:

- For $r = 1$ (i.e. Direct Send), the latency term reduces to: $\alpha(p-1)$.
- For all $k_i$ equal to 2 (i.e. Binary Swap), the latency term reduces to $\alpha\log_2(p)$.
- Depending on the $k_i$ values, the latency term is between Binary Swap (best case) and Direct Send (worst case).

The latency term of the Parallel Pipeline and Shift-Based algorithms is $\alpha(p-1)$, since there are $(p-1)$ stages of communication. The communication term is bounded by $\beta(p-1)(n/p)$, since each message contains $(n/p)$ pixels.

Table 1 summarizes the communication and computation costs of the different algorithms. We do not specify the computation term, but rather whether the computations can be overlapped with the communications for the given algorithm. Except for the 2-3 Swap algorithm, all algorithms are equivalent in terms of communications. They only differ on the latency term and on their capability to overlap computations with communications.

At that point of the analysis, we are interested in the impact of the latency term of the parallel image compositing algorithms for modern high performance, low latency interconnects. For instance, for a 4x DDR (Double Data Rate) IB network, the latency and transmission time can be measured as: $\alpha = 1\times10^{-6}$ s and $\beta = 0.53\times10^{-9}$ s (corresponding to 1800 MB/s): Figure 2 plots the evolution of the communication time for increasing numbers of processing nodes for the Binary Swap algorithm (best latency case) and the Direct Send, Parallel Pipeline and Shift-Based algorithms (worst latency case) for a 4096x4096x16 image.

| Algorithm | Latency | Comm. | Overlap. Blending |
|---|---|---|---|
| Direct Send | $\alpha(p-1)$ | $\beta n\frac{p-1}{p}$ | Yes |
| Binary Swap | $\alpha\log_2(p)$ | $\beta n\frac{p-1}{p}$ | No |
| 2-3 Swap | $4\alpha\log_2(p)$ | $1.3\beta n\frac{p-1}{p}$ | No |
| Radix-k | $\alpha\sum_{i=1}^{i=r}(k_i-1)$ | $\beta n\frac{p-1}{p}$ | Partially |
| Parallel Pipeline | $\alpha(p-1)$ | $\beta n\frac{p-1}{p}$ | No |
| Shift-Based | $\alpha(p-1)$ | $\beta n\frac{p-1}{p}$ | Yes |

**Table 1:** *Communication and computation costs of the parallel image compositing algorithms.*

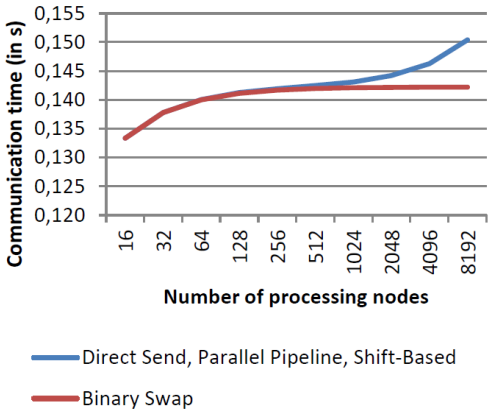


**Figure 2:** *Theoretical evolution of the communication time for increasing numbers of processing nodes on an ideal DDR IB crossbar for compositing of a 4096x4096x16 image.*

The latency term of the image compositing algorithms does not have a big impact on the total network time, except for very high processor counts (several thousands).

However, we recall that only the Parallel Pipeline and Shift-Based algorithms can benefit from the full cross-bisectional bandwidth provided by IB fat-trees. The Direct Send and Radix-k algorithms all suffer from hot spots on large IB fat-trees, and would not benefit from the full 1800 MB/s provided by DDR IB. Figure 3 plots the evolution of the communication time we may observe when running the different algorithms on real fat-tree IB clusters (we anticipate a performance degradation of 30%, which is a low estimate as compared to real runs).

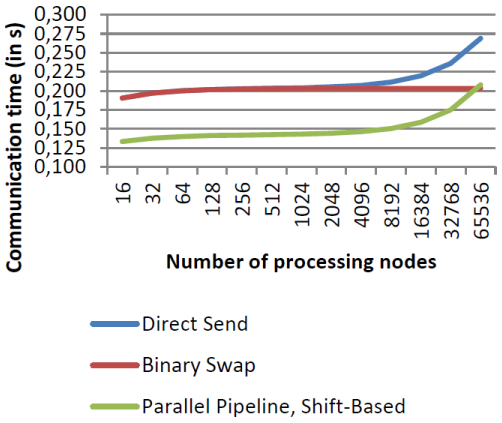


**Figure 3:** *Theoretical evolution of the communication time for increasing numbers of processing nodes on a real DDR IB fat-tree for compositing of a 4096x4096x16 image.*

The main advantage of our Shift-Based algorithm over the Binary Swap algorithm is its ability to overlap computations with communications. This has indeed a huge impact on the overall performance of the image compositing algorithm. On the other hand, it could be demonstrated that a routing scheme (different from the Fat-tree Unicast routing) could be computed to provide contention-free communication for the Binary Swap communication pattern.

In terms of scalability of the communication time on large IB fat-trees, the Parallel Pipeline and the Shift-Based algorithms are completely equivalent. The last, but not least, difference between the two algorithms is that the Parallel Pipeline does not permit to overlap computations with communications (similarly to the Binary Swap algorithm), while our Shift-Based algorithm does. We can conclude from our analysis that the Shift-Based algorithm is the most scalable on IB fat-trees with Fat-tree Unicast routing enabled.

## 5. Results

Our experimentations have been conducted on a 144-node fat-tree DDR IB cluster, running Linux CentOS 64-bit. Each node is composed of a single quad-core Intel Xeon X3440, and the nodes are interconnected with 24-port DDR switches on a 12-ary 2-tree topology. The OpenSM has been configured to run the Fat-tree Unicast routing engine.

We used the methodology and tools of Moody [Moo09] to measure the influence of contention. During the execution of the Shift-Based algorithm on the cluster, each process measures the time it takes to send its data to every other pair process, and computes the associated bandwidth. A 2D matrix is built from these collected values, where each element at row a and column b represents the send bandwidth process Pa measured when it sent to process Pb. The diagonal elements of this matrix represent a task sending to itself, which is not measured. The minimum, average and maximum values of the matrix are computed. The 2D matrix is then converted into a gray-scale bitmap image by representing each entry as a pixel. The maximum bandwidth of the matrix is set to a pixel value of 255 (white), while all other values are scales to a pixel value of 0 (black) depending on their percentage of the maximum bandwidth. A histogram of the pixel values is also computed and shown next to each gray-scale image. The horizontal axis plots the pixel value (from 0/black on the left to 255/white on the right), and the vertical axis plots the number of pixels of a given value in the image.

### 5.1. Performance on a Crossbar Switch

Figure 4 shows a run of our Shift-Based algorithm on 12 nodes connected on a single 24-port crossbar switch, with an image size of 4096x2048x16 bytes. The average bandwidth is 1827 MB/s and the final frame rate is 14.5 FPS. It can be easily concluded that there is very few contention, and that our algorithm benefits from the full cross-bisectional bandwidth of the switch.

**Figure 4:** *Shift-Based run on 12 nodes on a single 24-port crossbar switch (image size: 4096x2048x16 bytes). Final frame rate is 14.5 FPS.*

As a point of comparison, our implementation of the Direct Send algorithm on the same 12 nodes on the same single 24-port crossbar switch leads to a frame rate of 6.4 FPS.

### 5.2. Performance on IB Fat-Trees

Figure 5 shows a run of our Shift-Based algorithm on 144 nodes with Fat-tree Unicast routing enabled, with an image size of 4096x2048x16 bytes. The 144 nodes have been chosen in the exact same order given by the OpenSM routing engine. The average bandwidth is 1431 MB/s (that is 78% of the cross-bisectional bandwidth) and the final frame rate is 11.5 FPS.

In order to validate the effects of the routing mechanism on the performance of our Shift-Based algorithm, we performed the same run, this time with the 144 nodes listed in alphabetical order (that is not in the order given by the OpenSM routing engine). Figure 6 shows the result of this

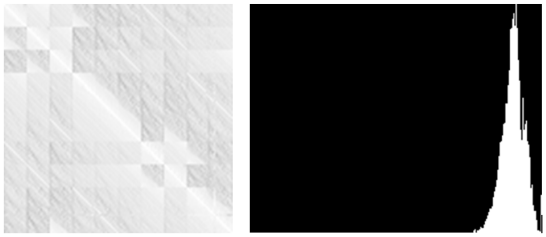

**Figure 5:** *Shift-Based run on 144 nodes with the Fat-tree Unicast routing engine (image size: 4096x2048x16 bytes). Final frame rate is 11.5 FPS. Nodes are taken in the order given by the OpenSM routing engine.*

run: the average bandwidth is 924 MB/s (that is 50% of the cross-bisectional bandwidth) and the final frame rate is 6.5 FPS. The gray-scale image and the associated histogram highlight the high level of link contention that occurred during this run.

As a point of comparison, our implementation of the Direct Send algorithm on the same 144 nodes leads to a frame rate of 4.7 FPS, whether we use the order given by the OpenSM routing engine or the alphabetical order.

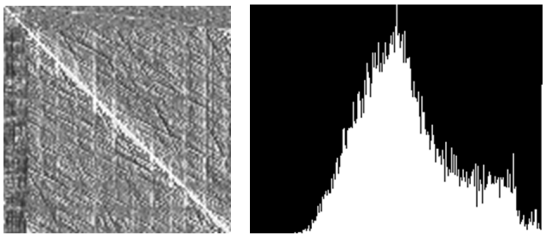

**Figure 6:** *Shift-Based run on 144 nodes with the Fat-tree Unicast routing engine (image size: 4096x2048x16 bytes). Final frame rate is 6.5 FPS. Nodes are taken in the alphabetical order.*

We also wanted to show the influence of the size of the data transferred over the network. Figure 7 shows a run of our Shift-Based algorithm on 144 nodes with Fat-tree Unicast routing enabled, with an image size of 4096x4096x16 bytes. The 144 nodes have been chosen in the exact same order given by the OpenSM routing engine. The average bandwidth is 1568 MB/s (that is 85% of the cross-bisectional bandwidth) and the final frame rate is 5.8 FPS. As compared to Figure 5 (for which we used a 4096x2048x16 image size), the gray-scale image is much clearer, showing even less contention on the network, and a better utilization of the cross-bisectional bandwidth.

As a final experiment, we have run our Shift-Based algorithm using all nodes counts from 12 to 144 of the 144 nodes with Fat-tree Unicast routing enabled, with several

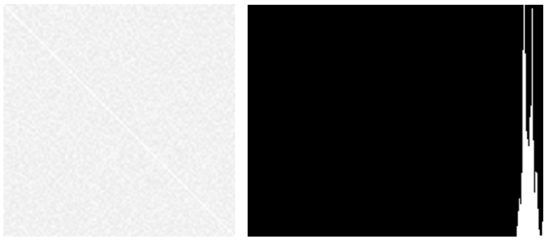

**Figure 7:** *Shift-Based run on 144 nodes with the Fat-tree Unicast routing engine (image size: 4096x4096x16 bytes). Final frame rate is 5.8 FPS. Nodes are taken in the order given by the OpenSM routing engine.*

image sizes: 2048x2048x16 bytes, 4096x2048x16 bytes and 4096x4096x16 bytes. Figure 8 shows the evolution of the final frame rate as a function of the number of nodes used. The nodes are taken in the order given by the OpenSM routing engine. We can see that the obtained final frame rate is consistent with the size of the image being composed (the 2048x2048x16 bytes image is four times faster to be composed as compared to the 4096x4096x16 bytes image) and stable with increasing numbers of processing nodes (we do not see any random and unexplained variation). It is important to note that the obtained results are fully reproducible from one run to another. This can be explained by the predictable shift communication pattern and the absence of hot spots made possible by the Fat-tree Unicast routing engine. The only unanticipated behavior is the presence of peak performance for counts of nodes that are multiple of 12, where 12 is half of the arity of the switches used in our fat-tree topology (i.e. the number of leaf processing nodes connected to the first level switches). These peaks can be explained by the fact that when we do not use a fully populated level-1 switch, some links are missing to provide contention-free routing for our shift permutation, leading to hot spots on the used links.

Figure 9 shows a run of our Shift-Based algorithm on 64 nodes with Fat-tree Unicast routing enabled, with an image size of 4096x2048x16 bytes. The 64 nodes have been chosen in the exact same order given by the OpenSM routing engine. The average bandwidth is 1159 MB/s (that is 63% of the cross-bisectional bandwidth) and the final frame rate is 8 FPS. The gray-scale image and the associated histogram highlight many hot spots in our shift permutation pattern.

As a point of comparison, our implementation of the Direct Send algorithm on the same 64 nodes leads to a frame rate of 5.2 FPS.

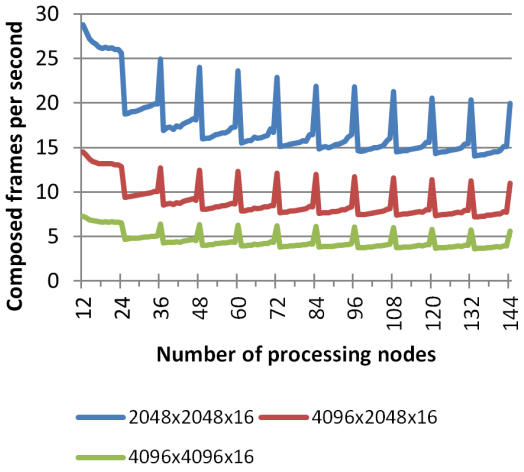


**Figure 8:** *Evolution of the image compositing frame rate with increasing numbers of processing nodes (from 12 to 144, with a step of 1). Nodes are taken in the order given by the OpenSM routing engine.*

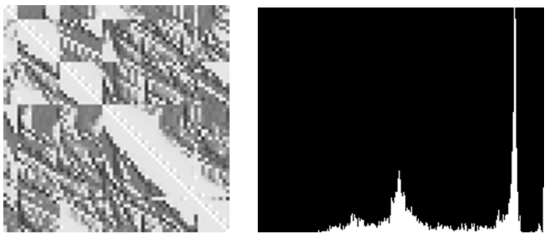

**Figure 9:** *Shift-Based run on 64 nodes with the Fat-tree Unicast routing engine (image size: 4096x2048x16 bytes). Final frame rate is 8 FPS. Nodes are taken in the order given by the OpenSM routing engine.*

## 6. Summary

### 6.1. Conclusion

In this paper, we have explored parallel image compositing algorithms on a growing class of supercomputers: the fat-tree IB clusters. We have built upon an in-depth study of the fat-tree IB topology on one hand and of existing parallel image compositing algorithms on the other hand, to introduce Shift-Based, a new algorithm based on the well-known shift permutation communication pattern. Our Shift-Based algorithm combines all the advantages of previous image compositing algorithms, and allows benefiting from a big part of the cross-bisectional bandwidth made available by IB fat-trees. We have proven that it is the most scalable algorithm for this kind of topology, and we have demonstrated its scalability on a 144-node fat-tree IB cluster with Fat-tree Unicast routing enabled. This cluster is the biggest one we have been able to have access to so far. However, we are confident that our conclusions can be extended to the case of much larger IB fat-trees, as long as a routing mechanism, such as the d-mod-k routing [Zah10], can be found to provide non-blocking traffic for shit permutations. Furthermore, we claim that the difference will be even bigger at higher processor counts, due to the increasing number of network contention for the other algorithms not relying on a shift permutation.

### 6.2. Future Work

The research work presented in this paper tends to prove that it is definitely worth the effort to take into account the network topology and its associated routing mechanisms in order to build efficient and scalable parallel algorithms. In the future, we plan to make similar investigations for parallel image compositing on other types of supercomputer architectures and to extend this work to other kinds of parallel algorithms.

## 7. Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see http://www.grid5000.fr).

## References

[CHPvdG07] CHAN E., HEIMLICH M., PURKAYASTHA A., VAN DE GEIJN R.: Collective communication: theory, practice, and experience. *Concurr. Comput. : Pract. Exper. 19* (September 2007), 1749–1783. doi:10.1002/cpe.v19:13. 5

[CMF05] CAVIN X., MION C., FILBOIS A.: Cots cluster-based sort-last rendering: performance evaluation and pipelined implementation. In *Visualization, 2005. VIS 05. IEEE* (oct. 2005), pp. 111 – 118. doi:10.1109/VISUAL.2005.1532785. 4

[HSL08] HOEFLER T., SCHNEIDER T., LUMSDAINE A.: Multistage switches are not crossbars: Effects of static routing in high-performance networks. In *Cluster Computing, 2008 IEEE International Conference on* (29 2008-oct. 1 2008), pp. 116 –125. doi:10.1109/CLUSTR.2008.4663762. 2, 3

[IBT04] IBTA - INFINIBAND TRADE ASSOCIATION: *IBTA specification 1.2, vol 1*. Tech. rep., 2004. URL: http://www.infinibandta.org/specs/. 2

[KLJ08] KERBYSON D. J., LANG M., JOHNSON G.: Infiniband routing table optimizations for scientific applications. *Parallel Processing Letters 18* (2008), 589–608. doi:10.1142/S0129626408003582. 4

[Lei85] LEISERSON C. E.: Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput. 34* (October 1985), 892–901. URL: http://dl.acm.org/citation.cfm?id=4492.4495. 3

[LRN96] LEE T.-Y., RAGHAVENDRA C., NICHOLAS J.: Image composition schemes for sort-last polygon rendering on 2d mesh multicomputers. *Visualization and Computer Graphics, IEEE Transactions on 2*, 3 (sep 1996), 202 –217. doi:10.1109/2945.537304. 1, 4

[MCEF94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H.: A sorting classification of parallel rendering. *Computer Graphics and Applications, IEEE 14*, 4 (jul 1994), 23 –32. doi:10.1109/38.291528. 1

[MKPH11] MORELAND K., KENDALL W., PETERKA T., HUANG J.: An image compositing solution at scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 25:1–25:10. doi:10.1145/2063384.2063417. 1

[Moo09] MOODY A.: *Contention-free routing for shift-based communication in MPI applications on large-scale InfiniBand clusters*. Tech. Rep. LLNL Report #LLNL-TR-418522, oct 2009. 7

[MPHK94] MA K.-L., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel volume rendering using binary-swap compositing. *IEEE Comput. Graph. Appl. 14* (July 1994), 59–68. doi:10.1109/38.291532. 1, 4

[Neu94] NEUMANN U.: Communication costs for parallel volume-rendering algorithms. *Computer Graphics and Applications, IEEE 14*, 4 (jul 1994), 49 –58. doi:10.1109/38.291531. 1, 4

[OIDK95] OHRING S., IBEL M., DAS S., KUMAR M.: On generalized fat trees. In *Parallel Processing Symposium, 1995. Proceedings., 9th International* (apr 1995), pp. 37 –44. doi:10.1109/IPPS.1995.395911. 3

[PGR*09] PETERKA T., GOODELL D., ROSS R., SHEN H.-W., THAKUR R.: A configurable algorithm for parallel image-compositing applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (New York, NY, USA, 2009), SC '09, ACM, pp. 4:1–4:10. doi:10.1145/1654059.1654064. 1, 4, 5, 6

[PV97] PETRINI F., VANNESCHI M.: k-ary n-trees: high performance networks for massively parallel architectures. In *Parallel Processing Symposium, 1997. Proceedings., 11th International* (apr 1997), pp. 87 –93. doi:10.1109/IPPS.1997.580853. 3

[YWM08] YU H., WANG C., MA K.-L.: Massively parallel volume rendering using 2-3 swap image compositing. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for* (nov. 2008), pp. 1 –11. doi:10.1109/SC.2008.5219060. 1, 4, 6

[Zah10] ZAHAVI E.: *D-Mod-K routing providing non-blocking traffic for shift permutations on real life fat trees*. Tech. Rep. CCIT Report #776. Technion, 2010. 3, 4, 9

[ZJKL10] ZAHAVI E., JOHNSON G., KERBYSON D. J., LANG M.: Optimized infiniband fat-tree routing for shift all-to-all communication patterns. *Concurr. Comput. : Pract. Exper. 22* (February 2010), 217–231. doi:10.1002/cpe.v22:2. 3, 4