Optimized Routing for Large-Scale InfiniBand Networks

Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine
Open Systems Lab,
Indiana University,
Bloomington IN 47405, USA
{htor,timoschn,lums}@cs.indiana.edu

***Abstract*—Point-to-point metrics, such as latency and bandwidth, are often used to characterize network performance with the consequent assumption that optimizing for these metrics is sufficient to improve parallel application performance. However, these metrics can only provide limited insight into application behavior because they do not fully account for effects, such as network congestion, that significantly influence overall network performance. Because many high-performance networks use deterministic oblivious routing, one such effect is the choice of routing algorithm. In this paper, we analyze and compare practical and theoretical aspects of different routing algorithms that are used in today's large-scale networks. We show that widely-used theoretical metrics, such as edge-forwarding index or bisection bandwidth, are not accurate predictors for average network bandwidth. Instead, we introduce an intuitive metric, which we call "effective bisection bandwidth" to characterize quality of different routing algorithms. We present a simple algorithm that globally balances routes and therefore improves the effective bandwidth of the network. Compared to the best algorithm in use today, our new algorithm shows an improvement in effective bisection bandwidth of 40% on a 724-endpoint InfiniBand cluster.**

I. Introduction

The focus of network performance assessment and improvement is often concentrated on the point-to-point metrics bandwidth and latency. Techniques such as the use of multiple transmission channels or operating system bypass are utilized to increase point-to-point communication performance. However, it is often the case that large improvements in overall bandwidth and latency can be achieved with optimized routing strategies. Practical tests [13] showed that some traffic patterns can suffer from up to 6.5 times lower bandwidth and 5 times higher latency in existing InfiniBand installations because of congestion.

A common measure to assess the performance of a complete network is *bisection bandwidth*: the bandwidth of a worst-case bisection of the physical network graph. This scheme ignores routing in the network and thus, networks that have full bisection bandwidth deliver only 55%–57% *effective* bandwidth for random bisections [13]. The *effective bisection bandwidth* (Λ, see Section IV-A) models the expected bandwidth if each network endpoint communicates with exactly one partner. The effective bisection bandwidth considers the routing strategy of the network and thus more accurately predicts parallel application performance.

Routing depends on many parameters, such as crossbar and buffer size. Additionally, practical constraints (e.g., purely deterministic destination-based route selection) often prohibit the use of excellent theoretical results in existing systems. In this work, we present a simple deterministic, fast and practical algorithm to generate oblivious routing tables. We demonstrate the practical applicability to large InfiniBand networks in comparison to all routing algorithms that are used by InfiniBand's Subnet Manager OpenSM.

Routing algorithms can be categorized in *oblivious* and *adaptive* algorithms. An oblivious algorithm determines a route for each source-destination pair (s, d) without considering the traffic on the network. An adaptive algorithm tries to adapt to the current traffic conditions in the network. Oblivious algorithms are attractive because they can be computed in advance, even if they are computationally expensive. An adaptive algorithm must be able to react quickly to changes in the (global) network and thus is often constrained to fast suboptimal algorithms. We note that optimal deterministic routing (with a minimal edge-forwarding index) in arbitrary networks is NP-hard [20]. Also, oblivious algorithms are usually unobtrusive (e.g., they do not need to monitor the network) and can often be implemented in a fully distributed manner. Online oblivious routing [4] combines oblivious and adaptive routing techniques and optimizes for a certain traffic pattern. However, reaction to changing input parameters like network topology and traffic demands happens slowly.

A routing algorithm must operate within the constraints of the underlying network architecture that limits several parameters, such as the number of logical paths in the network. In this work we use the InfiniBand Architecture (IBA), a commonly used high-performance interconnect, as an example. Several large-scale systems with more than 1000 endpoints use InfiniBand. The IBA does not mandate any topology, however, most deployed InfiniBand networks use a k-ary n-tree topology (e.g., fat tree [15] or Clos [7]). InfiniBand crossbars usually support virtual cut-through routing [14] with 24 full duplex links.

InfiniBand uses oblivious destination-based distributed routing and offers up to 128 virtual paths [25] between two endpoints with the Lid Mask Control (LMC) feature. More than

one path through the network might lead to more flexibility and alternative paths in case of network errors. Previous research [10] suggested that using multiple predetermined paths (either picking one randomly or round-robin) does not increase the average bandwidth. However, more elaborate schemes are clearly a base for future research. Another important feature of InfiniBand, the support of up to 15 virtual lanes, can be used to break routing cycles to avoid deadlocks in the network [16]. The destination-based routing mechanism limits the choice of algorithms because if two routes to the same endpoint lead through the same crossbar, then both must use the same path from that crossbar to the destination. This constrains algorithms that use distinct routes which can cross each other in the network (see Section III-B).

In this work, we focus on oblivious deterministic routing with a single route between any two endpoints. We define the two most important goals of our work: (1) minimize the path length between any two endpoints and (2) minimize the expected congestion (maximize the bandwidth). Minimal path lengths lead to smaller latencies under low load whereas minimal congestion leads to a higher throughput at medium and high load.

A. Background and Related Work

We model a physical network consisting of C crossbars and P endpoints as a connected graph $G = (V, E)$. The vertices $V = V_P \cup V_C$ can be split into V_P and V_C which are the sets of endpoints and crossbars in the graph respectively. Each endpoint V_P has exactly one neighboring crossbar V_C.

A complete routing R of a network consists of $P \cdot (P-1)$ paths among all ordered pairs u and $v \neq u$, $(u, v) \in V_P \times V_P$. A route from u to v, $r(u, v)$, is represented as a vector of $t+1$ vertices $r(u, v) = (r_0, r_1, \ldots, r_t)$ with $r_0 = u$, $r_t = v$, and t being the length of the path.

A route $r_m(u, v)$ is said to be minimal if it is an unweighted shortest path between u and v (i.e., spans a minimum number of vertices). A routing R_m is said to be minimal if all routes $r_m \in R_m$ are minimal. Minimal routes are important for maintaining minimal communication latency between the endpoints under low traffic.

To evaluate the routing behavior and network congestion situations, we define the number of routes leading though an edge $e \in E$ as its load $l(e)$ [6], [11].

The edge-forwarding index $\pi(G, R)$ of network G and routing R, introduced in [11], is defined as the maximum edge load in G

$$\pi(G, R) = \max_{\forall e \in E} l(e)$$

A communication pattern S is defined as a set of streams between the endpoints. The congestion of an edge (physical channel) can be derived by counting the number of streams that utilize routes that lead through the edge.

It seems intuitive that the edge-forwarding index defines the *balancedness* and thus the quality of the oblivious routing. However, we will show that this is not generally sufficient to model average bandwidths.

Another way to assess the quality of a routing in general networks would be to determine the worst-case pattern as described in [27]. Though, this method requires $\mathcal{O}(P^5)$ time and the worst-case pattern often fails to represent the common case. We argue that the *effective bisection bandwidth* Λ is a much more accurate measure for the quality of a set of routes R.

Many research groups have worked on optimal oblivious routing, but, most results are not directly applicable to InfiniBand. Räcke et al. [19] solved the congestion-minimization problem for arbitrary undirected networks close to optimal. Nevertheless, this algorithm needs to solve several NP-hard problems and requires randomization in the network which makes it impractical. Bienkowski [5] proposes a method that does not assign deterministic routes to sender/destination pairs, but solves a combinatorial multi-commodity flow (MCF) problem. Azar et al. [3] improved Räcke's work and propose a polynomial-time algorithm based on MCF with similar performance. However, this algorithm is based on linear programming and is also not suitable for large-scale systems due to its high computational demand [2]. In the MCF model, each crossbar only knows what fraction of the traffic has to use a certain edge in the graph. As this would require non-deterministic routing schemes, it is not applicable to InfiniBand. Towles et al. [28] discuss the trade-offs between minimal path length and high throughput on k-ary n-cube networks. They also use linear programming to solve the routing problem as a MCF-problem and end up with a probability distribution over the paths in the network for each source-destination pair.

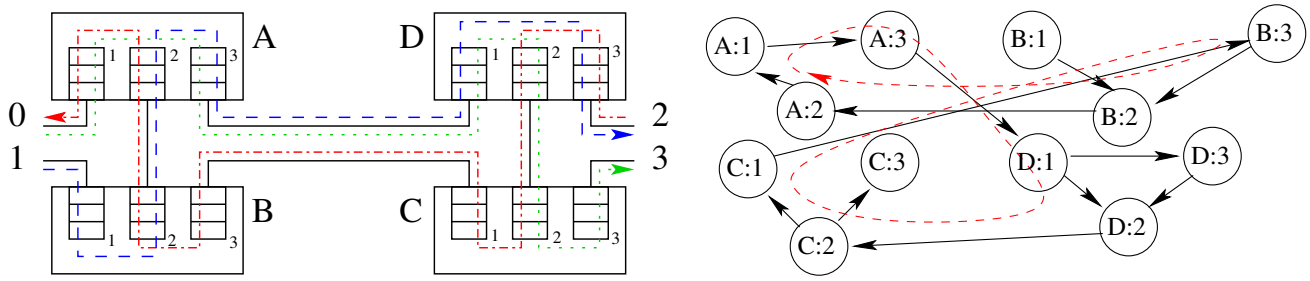
Many other schemes that are used in practice are bound to a particular network architecture, for example k-ary n-cubes or fat tree networks [29]. Generic algorithms do often not find good routes or are too computationally expensive to be used in practice [1].

In the following, we focus on practically used oblivious routing algorithms within the constraints of the InfiniBand Architecture.

II. Current Practical Routing Algorithms

InfiniBand uses oblivious destination-based distributed routing. Each switch has a lookup table (*Random-* or *Linear Forwarding Table*) that defines which port leads to which endpoint. On startup or after topology changes, the Subnet Manager (SM) discovers the topology, computes the forwarding tables for each switch, and uploads them to the switches. The current implementation, OpenSM, offers five different routing algorithms: MINHOP, UPDN, FTREE, DOR and LASH. Additionally, it can load the forwarding tables from a file.

MINHOP finds minimal paths among all endpoints and tries to balance the number of routes per link locally at each switch (under the constraint of minimal routes). However, this method can lead to circular dependencies among the switch buffers [8] which might lead to a deadlock of the network.



(a) Routes (b) Buffer Dependencies

Fig. 1. Credit loop in a network with four crossbars.

UPDN uses the Up*/Down* routing algorithm [22] to avoid circular buffer dependencies by restricting the number of possible paths in the network [21].

FTREE implements a routing scheme optimized for fat trees [29] which is also deadlock-free but requires a fat tree network topology.

DOR (Dimension Order Routing) routes along the dimensions of a k-ary n-cube to determine shortest paths [24] and might create circular buffer dependencies.

LASH routing [23] uses Virtual Lanes (VL) to break cyclic dependencies among channels of the underlying DOR scheme.

Refer to the OpenSM documentation for a detailed description of each routing algorithm.

A. Credit Loops and Deadlocks

In the design of routing algorithms, one has to be careful to avoid deadlock situations. Deadlocks are well known in wormhole routing schemes [8]. However, InfiniBand routing presents a slightly different problem. InfiniBand uses a credit-based flow control scheme in hardware. This means that each output port can only send packets if it has *credits* at the destination input port. This avoids packet loss at congested switches. Though, cyclic dependencies among input buffers can lead to a situation where all clients are waiting for credits and the network is effectively deadlocked. Those cyclic dependencies are called *credit loops* and should be avoided by the routing algorithm. Figure 1(a) shows a small network with four crossbar switches (A, B, C, D), four endpoints $(1, 2, 3, 4)$ and three routes $R = \{(2, D, C, B, A, 0), (0, A, D, C, 3), (1, B, A, D, 2)\}$. The crossbars have three ports each with input buffers to store three packets. A packet is only sent if an element in the input buffer at the destination is free. This makes the progression of the input buffer which has a route leading to another buffer dependent on the availability of a free slot. The full dependency graph for the buffers (ports) is shown in Figure 1(b). This graph has a cycle, which means that there is a potential for deadlock (depending on the injection pattern) in the system. This cycle is called *credit loop* in InfiniBand terms.

The above problem is similar to the well-known deadlock problem [8] of wormhole routing. One possible solution is implemented by the Up*/Down* algorithm that constrains the number of possible paths. Another method, which is used by the LASH algorithm, is to find a good routing for a network and then break the cycles by assigning different VLs to routes in the cycle. This is possible because each VL has its own logical buffers. Some OpenSM algorithms, like MINHOP and DOR simply ignore the issue that might lead to network instabilities. However, the necessary constellations for deadlocks happen rarely and deadlocks can be broken with packet timeouts (this method is not favorable).

III. Routing based on Shortest Paths

As discussed before, different strategies can be used to optimize oblivious routing. Though, most practical algorithms seem to optimize for path length and only insufficiently for throughput (e.g., with only local optimizations as MINHOP does). A routing algorithm also has to be fast so that it can be used in large-scale systems. Thus, we propose a new and simple routing strategy based on a single source shortest path (SSSP) algorithm (one possible implementation is Dijkstra's algorithm [9]). We also analyze two different greedy heuristics to minimize the edge-forwarding index.

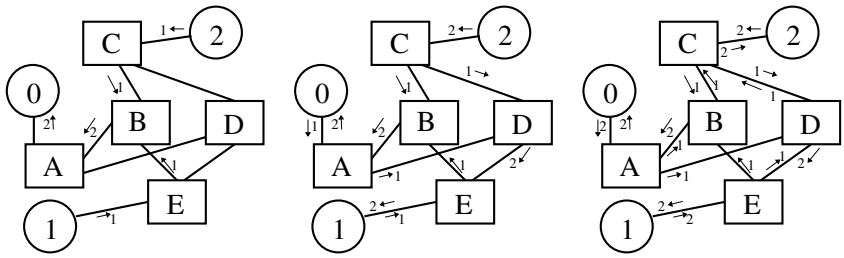
A. P-SSSP Routing

The first variant starts with the graph G and edge weights of all edges $e \in E$ are one. The algorithm iterates in unspecified order over all endpoints $u \in V_P$ and finds reverse shortest paths from u to all other endpoints $v \in V_P$. In undirected networks we use the single destination shortest path algorithm, which is equivalent to the reverse SSSP, to satisfy the constraints of oblivious destination-based routing. The tree structure of the shortest path tree automatically generates valid destination-based routes. After each endpoint, the algorithm replaces the edge weights with the updated number of routes that pass through each edge. The main difference between OpenSM's MINHOP and P-SSSP is that our algorithm performs a global optimization of the edge loads whereas MINHOP does it only locally at each switch.

The pseudo code for the algorithm is shown in Figure 2(a) and Figure 2(b)–2(d) show the three steps of the algorithm for routing from endpoint 0, 1, and 2. The figure is drawn as an undirected graph for readability, however, P-SSSP uses a directed graph. The routes determined on round 1, 2, and 3 are R_1={(1,E,B,A,0), (2,C,B,A,0)}, R_2={(0,A,D,E,1), (2,C,D,E,1)}, and R_3={(0,A,B,C,2), (1,E,B,C,2)} respectively.

Input: Network $G = (V_p \cup V_c, E)$
Output: Routes R
foreach $u \in V_p$ **do**
  comp. shortest paths from u to all $v \in V_p$
  add reverse paths to forwarding tables (R)
  update edge weights along paths

(a) The P-SSSP Algorithm



(b) From endpoint 0 (c) From endpoint 1 (d) From endpoint 2

Fig. 2. The P-SSSP algorithm and an example with three endpoints.

The final values at each edge, e, represent the edge loads $l(e)$. To determine the edge-forwarding index of the network, we must ignore edges leading to and coming from endpoints (in this network (0,A), (1,E), and (2,C)), because their edge load $l(e) = P - 1$ in all cases (they must connect to all $P - 1$ other endpoints). Thus, the edge forwarding index of our example network X is $\pi(X, R) = 2$ (the maximum edges are (D,E) and (B,A)).

This algorithm optimizes $\Lambda(G, R)$ by balancing the routes and minimizing $\pi(G, R)$. However, this optimization might not choose shortest paths between every pair of endpoints which might impact the latency under low load. To force the implementation to choose shortest paths (minimal routes), we initialize the edge weights not with one but rather with P^2 so that it is always cheaper to increase the utilization of an edge than to take a longer path.

B. P^2-SSSP Routing

It seems that the variant with P SSSP runs does not balance the edge-forwarding index ideally because it only considers and updates the edge weights P times. Thus, a more accurate greedy heuristic to minimize the edge-forwarding index would be to perform the SSSP for each source-destination pair and update the edge weights $P(P - 1)$ times (one for each pair of endpoints). The pseudo code for the algorithm is shown in Figure 3(a).

P^2-SSSP needs 6 rounds in our example network X and results in the routes R_1={(0,A,B,E,1)}, R_2={(0,A,D,C,2)}, R_3={(1,E,D,A,0)}, R_4={(1,E,B,C,2)}, R_5={(2,C,B,A,0)}, R_6={(2,C,D,E,1)}. The final edge weights are shown in Figure 3(b). The figure looks more balanced than Figure 2(d) from P-SSSP and has a lower forwarding index $\pi(X, R) = 1$. Figure 3(c) and Figure 3(d) give another example network $\hat{X}$ for a network routed with P-SSSP and P^2-SSSP and forwarding indexes $\pi(\hat{X}, R)$ of 3 and 2 respectively.

C. Computational Complexity

Dijkstra's SSSP algorithm can be implemented with a Fibonacci heap. With $m = |E|$ and $n = |V|$, it has a complexity of $\mathcal{O}(m+n \log n)$. The P-SSSP algorithm is called n times—once for each endpoint $u \in V_P$ and the P^2-SSSP algorithm is called n^2 times—once for each pair of endpoints $(u, v) \in V_P \times V_P$. Thus, the overall complexity of P-SSSP and P^2-SSSP are $\mathcal{O}(nm + n^2 \log n)$ and $\mathcal{O}(n^2 m + n^3 \log n)$ respectively. The complexity can further be reduced with linear-time SSSP algorithms specialized to integer edge weights [26].

D. Handling Credit Loops

P-SSSP and P^2-SSSP do not inherently prevent credit loops. This means that, as with MINHOP, a generated routing could contain credit loops. We note that, if shortest paths are enforced, P-SSSP routing is credit-loop free on some topologies, such as two-stage fat trees. However, the generated routes for other topologies like tori are not guaranteed to be free of credit loops. In this case, we propose to use a strategy similar to LASH [23], i.e., break the credit loops by splitting routes that contribute to a loop into different virtual lanes (VL). Skeie et al. argue in [23] that the total number of needed VLs grows sublinearly with the network size.

This technique can be used until the 15 VLs of InfiniBand are exhausted and the remaining endpoints can be routed using a less efficient but inherently deadlock free algorithm, for example Up*/Down*. Another workaround, which is used in many systems, is to ignore credit loops and set the packet timeout in the switches to break deadlocks. This is workable in practice because even if credit loops exist, the probability that all buffers run full and the network deadlocks is small (much smaller than in wormhole-routed systems) and the resulting deadlock can be resolved by packet timeouts.

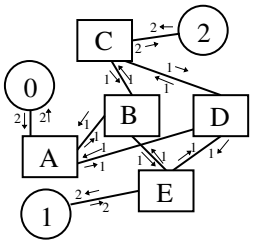
In the next section we show simulation results that compare the algorithms implemented in OpenSM and our SSSP-based algorithms for several different networks. We enforced minimal paths for P-SSSP and P^2-SSSP in all our examples such that the latencies are minimal like in MINHOP.

IV. Network Congestion Simulations

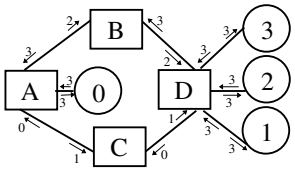
In order to compare the quality of different routing strategies, we simulate the bandwidth of random communication patterns in the network (we ignore the latency because our routing strategies are restricted to minimal routes R_m). To do this, we require a connected network G and routing R which contains a valid route between any two endpoints. We model an ideal flow control mechanism, i.e., each stream $s = (u, v)$ gets 100% of the bandwidth $\mathcal{B}$ divided by the maximum congestion along the path $r(u, v) \in R$.

```
Input: Network G = (V_p ∪ V_c, E)
Output: Routes R
foreach u ∈ V_p do
    foreach v ∈ V_p do
        comp. shortest path u → v
        /* the path is constrained by
        destination-based routing! */
        add path to forwarding tables (R)
        update edge weights along path
```

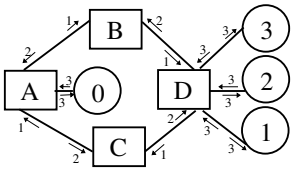
(a) The P^2-SSSP Algorithm



(b) P^2-SSSP Result



(c) P-SSSP Result



(d) P^2-SSSP Result

Fig. 3. Different networks with P^2-SSSP and P-SSSP routing.

A. Simulating Effective Bisection Bandwidth

The effective bisection bandwidth $\Lambda(G, R)$ of a network G is computed by simulating the congestion of N random *bisect* patterns. In a bisect pattern S_{bi}, all endpoints are divided into two subsets of size $P/2$ and each endpoint in the first subset sends to exactly one endpoint in the other subset (cf. perfect matching).

The bisection bandwidth of such a pattern is computed in two steps. In step one, each stream $(u, v) \in S_{bi}$ is routed along $r(u, v) \in R$ through the network and a congestion counter $\tau(e)$ of each traversed edge e (representing a physical link) is increased. The bandwidth of a stream $s = (u, v)$ is, according to our linear model, the link bandwidth $\mathcal{B}$ divided by the maximum congestion $\tau(e_i)$ along the path $r(u, v) = (e_1, \ldots, e_l)$. The second step computes the sum of all bandwidths along all streams:

$$\Gamma(S_{bi}) = \sum_{(u,v)\in S_{bi}} \frac{\mathcal{B}}{\max\{\tau(e_i)|e_i \in r(u, v)\}}$$

The effective bisection bandwidth $\Lambda(G, R)$ is now computed as the average of N different bisection bandwidths for random bisection patterns S_{bi}:

$$\Lambda(G, R) = \frac{1}{N} \sum_{N \text{ rand } S_{bi}} \Gamma(S_{bi})$$

The number N of different patterns should be large. Practically, only a very small fraction of the $\mathcal{O}(P!)$ search space can be covered. However, extensive simulations showed that the average bisection bandwidth with random bisect patterns stabilizes quickly [13]. We used $N = 10000$ and the Mersenne Twister algorithm [17] to generate random bisections for our simulations and benchmarks.

The effective bisection bandwidth $\Lambda(G, R)$ is often reported as a fraction of the full bisection bandwidth $\mathcal{B} \cdot P/2$. For example $\Lambda(G, R) = 1/2$ means that, on average, the network delivers half of the theoretical bandwidth if all endpoints communicate.

The edge-forwarding index can be computed in a similar way. Step one walks all $P(P-1)$ routes and increases a usage-counter for each visited edge. Step two simply records the maximum edge usage along each route.

It is obvious that for all edges $e \in E$ holds: $l(e) \leq \pi(G, R) \leq \tau(e)$. Thus, a routing r that reduces the edge-forwarding $\pi(G, R)$ also reduces the maximum possible congestion. However, it remains unclear how the average case is affected.

In the following sections, we simulate different network topologies and routings. We used OpenSM version 3.2.2 with the InfiniBand Simulator `ibsim` in order to compute the routes for our network topologies. OpenSM implements all algorithms that we discussed in Section II. Because the performance of the LASH algorithm was similar to DOR in all simulated results, we omitted it in several plots. The FTREE implementation did not run[1] on any of our real-world system inputs, thus, we only report FTREE results on artificially generated fat tree networks.

B. Real-World Systems

In this section we discuss the simulation results for several of today's largest InfiniBand-based clusters. We queried the network topology and routing tables of each system with the tools `ibdiagnet` and `ibnetdiscover`. The systems under consideration are:

a) Thunderbird: at Sandia National Laboratories is, with 4390 endpoints, the system with the biggest endpoint-count. It has a half-bisection bandwidth fat tree network.

b) Ranger: at Texas Advanced Computing Center is, with 4080 endpoints and 16 processing cores per endpoint, the fastest (peak floating-point performance) of the systems under consideration. It has two Sun Magnum 3456 port switches with five fat tree stages offering full bisection bandwidth.

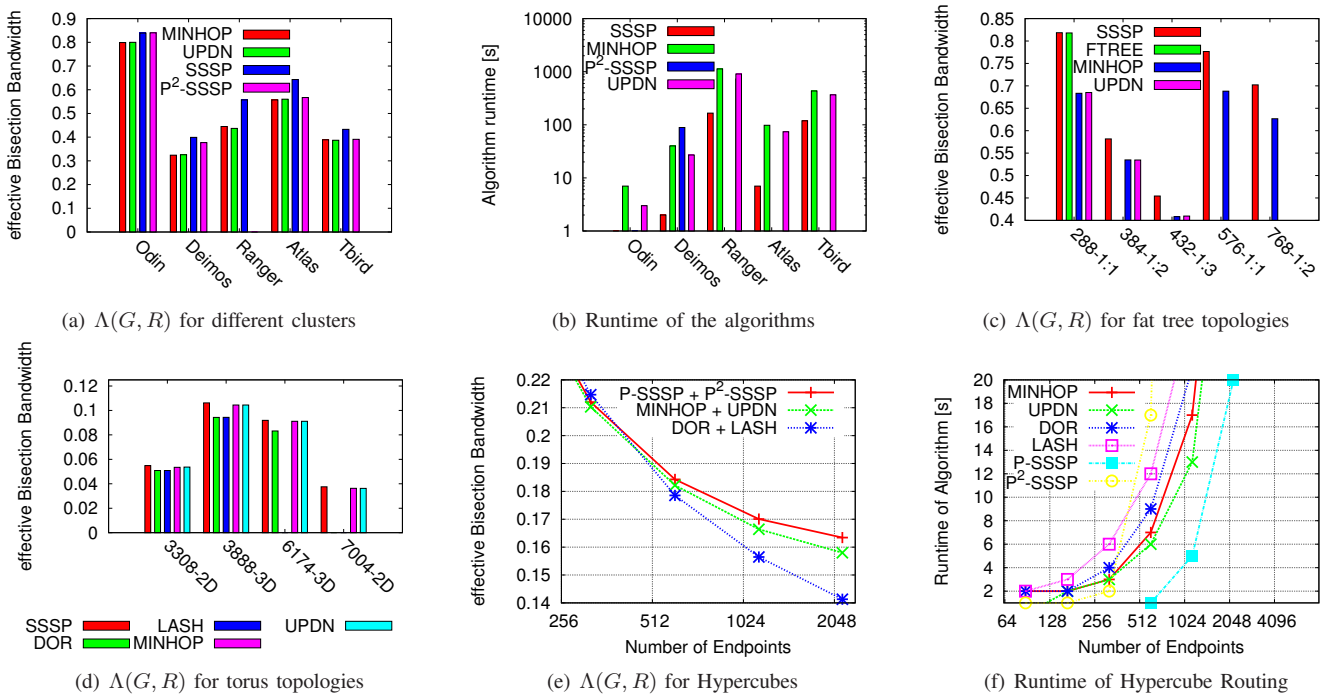
c) Atlas: at Lawrence Livermore National Laboratory has 1142 endpoints, connected with a full bisection bandwidth fat tree.

d) Deimos: at the Technical University of Dresden represents a mid-sized system with 724 endpoints. It consists of three 288-port switches, connected in a 30 link-wide chain (cf. Figure 7). We use this system for simulations and benchmarks.

e) Odin: at Indiana University is a small 128 endpoint cluster with a single switch (fat tree). We use this system for simulations as well as for benchmarks.

Figure 4(a) shows the simulated effective bisection bandwidth with different routing schemes on the real-world systems. We see that P-SSSP significantly improves the effective

[1] it seems that the implementation only accepts fully populated fat trees



(a) $\Lambda(G, R)$ for different clusters

(b) Runtime of the algorithms

(c) $\Lambda(G, R)$ for fat tree topologies

(d) $\Lambda(G, R)$ for torus topologies

(e) $\Lambda(G, R)$ for Hypercubes

(f) Runtime of Hypercube Routing

Fig. 4. Comparison of routing algorithms on different topologies.

bisection bandwidth for all systems by balancing the routes globally. However, P^2-SSSP performs worse, even though it balances the routes better and often finds a set of routes with a smaller edge-forwarding index. We will discuss this phenomenon later.

Figure 4(b) compares the running times of the different routing algorithms on the tested systems. It is important to mention that the runtimes for the algorithms in OpenSM represent runs with the InfiniBand simulator while the P-SSSP and P^2-SSSP implementations are simply implemented in C++. This means that quantitative comparisons are not possible, but trends can be assessed from those graphs. We see that P-SSSP has a similar scaling behavior to the OpenSM algorithms. P^2-SSSP does not scale as well and did, like LASH, not terminate after several hours with the biggest system inputs.

In the next section, we analyze the performance of the routing algorithms for different topologies supported by special algorithms in OpenSM.

C. Fat Tree Networks

We investigate the different routing strategies on generated ideal fat tree networks. We consider fat trees based on 24-port crossbars with different bisection bandwidths. Figure 4(c) shows the results for fat trees of different sizes. The bisection bandwidth is indicated by the oversubscription ratio, e.g., "1:2" means half bisection bandwidth. We see that the P-SSSP routing is able to increase the effective bisection bandwidth significantly comparable to the fat-tree optimized routing. The FTREE algorithm did not run on the fat tree networks with more than two levels[2].

D. k-ary n-cube Networks

In this section we show results for artificially generated hypercube (2-ary n-cube) networks and 2D/3D torus (k-ary 2/3-cube) networks constructed with 24-port InfiniBand crossbars. We assume that the DOR algorithm delivers better results on those topologies.

Figure 4(d) shows that the routing for different Torus networks can also be enhanced with the P-SSSP algorithm. However, the gain is rather small because the local decisions made by MINHOP are similarly good on those topologies.

Figure 4(e) shows the simulation results for hypercube topologies. We see also a minimal increase in effective bisection bandwidth. We see that the P-SSSP is again performing best, even better than the DOR and LASH algorithms which are optimized for the network type. This is again due to the better balancing of routes.

Figure 4(f) shows the runtime of the routing algorithms. P^2-SSSP quickly passes everything and is not feasible for large-scale networks. The P-SSSP implementation is again scaling similar to the other algorithms. However, we conclude that none of today's routing algorithms scales well to larger endpoint-counts (> 4096).

E. Random Networks

We also investigate randomly connected networks as examples for grown infrastructures. We generate random networks

[2]it aborted with a segmentation violation

with relatively high bandwidth by using 24-port crossbars with 12 endpoints attached to each of them. The remaining 12 ports of each switch are connected randomly to other crossbars. Figure 5 shows the performance for eight randomly generated networks with 3000 and 516 endpoints. Again, the P-SSSP algorithm performs better than all others even though the difference is not as significant as in the other examples. This is due to the high connectivity in our random networks. We expect a bigger difference if we add more endpoints per switch.

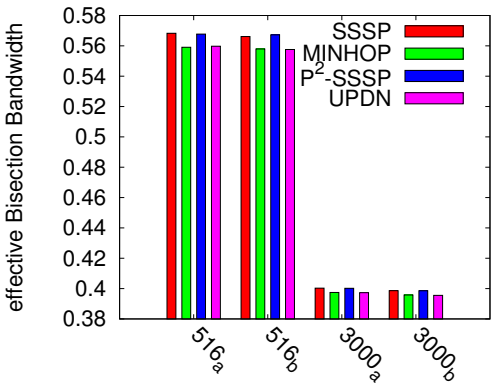


Fig. 5. $\Lambda(G, R)$ for Random Topologies.

The most important question now is why P^2-SSSP has a lower effective bisection bandwidth than P-SSSP even though it lowers the edge-forwarding index.

F. Influence of the Edge-Forwarding Index

Simulations show that the P-SSSP algorithm performs better than the P^2-SSSP algorithm even though the edge-forwarding index of the generated routes is smaller in the latter case. We note that the edge forwarding index only provides an upper bound for the congestion but does not allow any reasoning about average bandwidths or the effective bisection bandwidth. Thus, we argue that even though the forwarding index has long been used as a metric to asses routes [6], its influence on practical communication patterns might be low.

We discuss this with an example network, shown in Figure 6, where P^2-SSSP computes a routing with a lower forwarding index (and lower edge loads) than P-SSSP , but the effective bisection bandwidth is also lower.

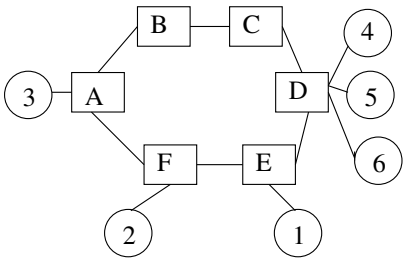


Fig. 6. Example Topology for P vs P^2-SSSP .

We only discuss the parts of the routing that are significant to the problem. We assume that the algorithms discover routes in numerical order, i.e., P-SSSP generates all routes to endpoints $1, 2, 3, \ldots, 6$ in this order and the P^2-SSSP generates the routes $(1, 2), (1, 3), \ldots, (1, 6), (2, 0), (2, 1), \ldots, (6, 5)$. P-SSSP and P^2-SSSP would place all routes between 1 and $4, 5, 6$ and between 2 and $4, 5, 6$ on edge (E,D) (if we constrain it to minimal path-lengths). Now, when P^2-SSSP tries to find all routes from $4, 5, 6$ to 3, both algorithms use different approaches. P-SSSP uses the upper path (D,C,B,A) for all routes while P^2-SSSP balances the routes between upper and lower path (remember that the graph is directed, so the load on edge (D,E) is initially zero). The problem is that this balancing of paths having a single destination is not beneficial for bisect patterns (and most application patterns) because the single destination will be in at most one pair. However, the balanced routes could lead to a congestion on edge (F,E) if the lower path is taken for the pair $(6, 3)$ and 1 and 2 are communicating. This does not happen in the P-SSSP case. Thus, this difference is mostly an artifact of the greedy heuristic.

V. Benchmarking Real-World Systems

We implemented our new routing scheme with the file-based routing in OpenSM in order to show the practical benefits for real-world clusters. We benchmarked and compared the effective bisection bandwidth of different routing strategies on two systems, the 724-endpoint Deimos cluster at Technical University of Dresden and the 128-endpoint Odin cluster at Indiana University. Deimos is equipped with PCIe 1.1 HCAs which deliver a point-to-point bandwidth of 946 MiB/s. The Odin system has older PCI-X HCAs which deliver 584 MiB/s (only $\approx 60\%$ of the InfiniBand line rate). Thus, we expect to see the influence of congestion very clearly on Deimos while the influence of congestion on Odin is expected to be small (as a congestion of two will have nearly no performance impact).

Odin is connected to a single 144 port InfiniBand switch which is an internal fat tree with 12 leaf-switches and full bisection bandwidth. Deimos consists of a chain of three 288 port switches, which are internal fat tree networks with 24 leaf switches, connected with 30 cross-cables each resulting in a rather low bisection bandwidth. The network topology of Deimos is shown in Figure 7.

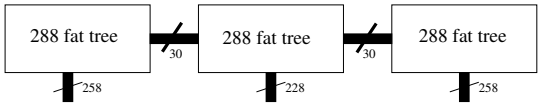
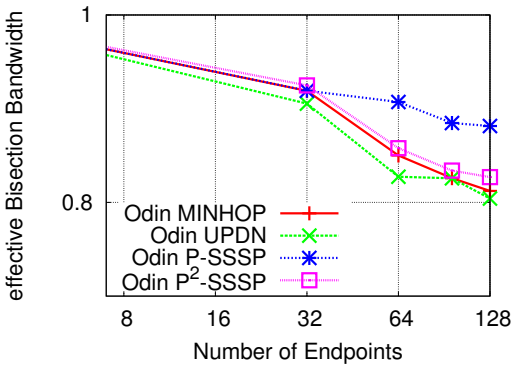
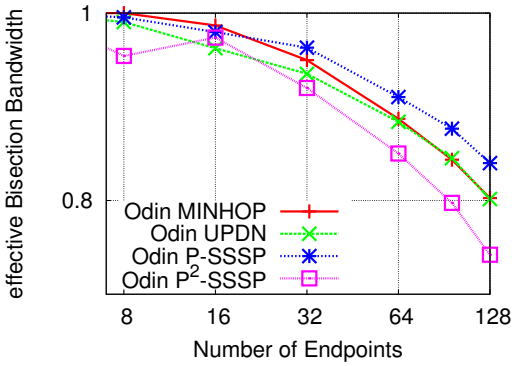


Fig. 7. Deimos Network Topology.

In our benchmark, we first generate a random bisection pattern for all processes in the communicator (we started all jobs with one process per endpoint). The bisection generation on rank 0 is performed as described in Section IV-A. Rank 0 distributes the pairing information to all other processes. Then, all pairs of processes synchronously start to exchange 50 messages for size 1 MiB (bidirectional) and measure the

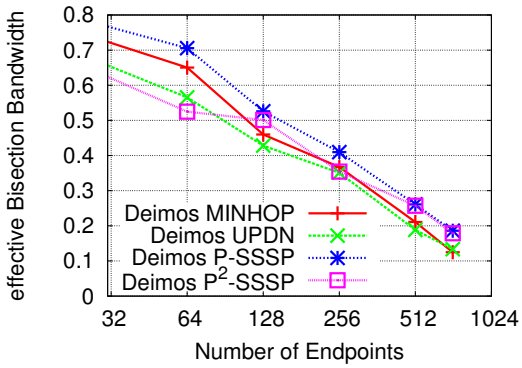


(a) Odin benchmark

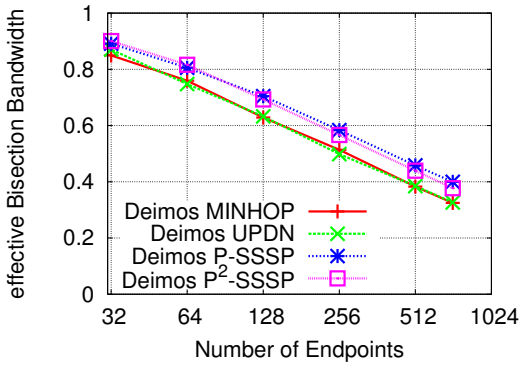


(b) Odin simulation

Fig. 8. Measured and simulated effective bisection bandwidths on Odin.



(a) Deimos benchmark



(b) Deimos simulation

Fig. 9. Measured and simulated effective bisection bandwidths on Deimos.

achieved bandwidth. This procedure is repeated with 1000 random mappings for each communicator size and the average result (an estimation of $\Lambda(G, R)$) is reported. The benchmark is implemented in the open-source measurement tool Netgauge [12].

Figure 8(a) shows the benchmark results on the Odin system. The effective bisection bandwidths are rather high because the endpoints can not utilize the full network speed and are thus less affected by congestion. However, we see that the bandwidth is decreasing with an increasing number of endpoints. We also clearly see the benefits of the new P-SSSP routing, which improves the benchmarked effective bisection bandwidth by 18% with regards to the best OpenSM routing at full scale. We also see that the simulation in Figure 8(b) shows correct trends for all routing algorithms. The absolute values are slightly different due to various effects in the network (i.e., flow control [18] and the slow PCI-X link).

Benchmark results for the effective bisection bandwidth on the 724-endpoint Deimos system are shown in Figure 9(a). Those results also show that our new routing strategy outperforms all routing algorithms in OpenSM significantly while maintaining minimal-length routes. The relative performance improvement increased with the number of endpoints up to 40% on 724 endpoints. Thus, we conjecture that the gain for larger networks would be even higher.

Figure 9(b) shows the simulation results on the Deimos system. The simulated bandwidths are higher than the real bandwidths because Deimos has, as opposed to Odin, full bandwidth at the endpoints. The difference between simulation and benchmark can be explained with effects in flow-control schemes, which can reduce the throughput to 60–75% [18].

VI. Conclusions and Future Work

We proposed two simple oblivious routing algorithms for networks with distributed destination-based routing (e.g., InfiniBand). Both algorithms use a single-source shortest pairs algorithm to minimize the edge-forwarding index (the maximum number of routes per edge) of the network. Compared to other implementations, such as OpenSM's MINHOP, optimization in our approach is done globally instead of locally at each switch. Thus, the proposed algorithms achieve a better balance of routes in the network.

We simulated the congestion and effective bisection bandwidth of all algorithms that are implemented in OpenSM on different real-world and artificial network topologies. The simulations showed that the P^2-SSSP algorithm, which balances the edge loads better and thus finds a routing with a lower edge-forwarding index, performs slightly worse than the P-SSSP algorithm. This shows that while the edge-forwarding index serves well as a theoretical lower bound to the minimal point-to-point bandwidth in the networks, it is not

necessarily a good predictor for effective (average) bisection bandwidth. However, we can clearly see that our algorithms (which are based on a heuristic that strives to minimize the edge-forwarding index) improve the balance of routes and the effective bisection bandwidth significantly relative to the routing schemes in OpenSM.

We showed that our routing improves the effective bisection bandwidth up to 11% on the 4390 endpoint Thunderbird cluster, 25% on the 4080 endpoint Ranger cluster and 15% on the 1142 endpoint Atlas cluster. We used the OpenSM file-based routing method to deploy our routing scheme on the 128 endpoint Odin cluster and the 724 endpoint Deimos system. The benchmarked effective bisection bandwidth on those systems could be increased by 18% on Odin and 40% on Deimos. As discussed in Section IV-A, our linear congestion model underestimates the effects of congestion. For example, the simulation predicted only an improvement of 23% for Deimos and 5% for Odin. Thus, it is reasonable to also expect higher practical gains on Ranger and Atlas than the simulation predicts.

In future works, we are trying to improve routing to support different (collective) patterns efficiently while retaining the high effective bisection bandwidth. We are also investigating better mapping strategies and collective communication algorithms optimized to a particular set of routes.

Acknowledgments

First we thank Guido Juckeland and Michael Kluge from the Technical University of Dresden for providing us access to and technical support with the Deimos cluster system. Simon Wunderlich, Jeremiah Willcock, Tor Skeie, and Sven Eckelmann provided helpful comments. Thanks to Adam Moody and Ira Weiny (LLNL) who provided the Atlas system topology, Christopher Maestas (Sandia National Labs) who provided the input file for the Thunderbird cluster, and Len Wisniewski (Sun) and the TACC who provided the Ranger input. Many computations have been performed on the CHiC cluster in Chemnitz. This work was partially supported by a grant from the Lilly Endowment, National Science Foundation grant EIA-0202048, and a gift from the Silicon Valley Community Foundation on behalf of the Cisco Collaborative Research Initiative.

REFERENCES

[1] Francisco José Alfaro, Aurelio Bermúdez, Rafael Casado, José Duato, Francisco J. Quiles, and José L. Sánchez. On the Performance of Up*/Down* Routing. In *CANPC '00: Proceedings of the 4th International Workshop on Network-Based Parallel Computing*, pages 61–72, London, UK, 2000. Springer-Verlag.

[2] David Applegate and Edith Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 313–324, New York, NY, USA, 2003. ACM.

[3] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. Optimal oblivious routing in polynomial time. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 383–388, New York, NY, USA, 2003. ACM.

[4] Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Online oblivious routing. In *In Proceedings of SPAA'03*, pages 44–49.

[5] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 24–33, New York, NY, USA, 2003.

[6] F. R. K. Chung, E.G. Coffman, M.I. Reiman, and B. Simon. The forwarding index of communication networks. *IEEE Trans. Inf. Theor.*, 33(2):224–232, 1987.

[7] C. Clos. A study of non-blocking switching networks. *Bell System Technology Journal*, 32:406–424, 1953.

[8] William J. Dally and Charles L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. pages 345–351, 1994.

[9] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(269–270):6, 1959.

[10] P. Geoffray and T. Hoefler. Adaptive Routing Strategies for Modern High Performance Networks. In *16th Annual IEEE Symposium on High Performance Interconnects (HOTI 2008)*, pages 165–172. IEEE Computer Society, Aug. 2008.

[11] M. C. Heydemann, J.C. Meyer, and D. Sotteau. On forwarding indices of networks. *Discrete Appl. Math.*, 23(2):103–123, 1989.

[12] T. Hoefler, T. Mehlan, A. Lumsdaine, and W. Rehm. Netgauge: A Network Performance Measurement Framework. In *High Performance Computing and Communications, Third International Conference, HPCC 2007, Houston, USA, September 26-28, 2007, Proceedings*, volume 4782, pages 659–671. Springer, 9 2007.

[13] T. Hoefler, T. Schneider, and A. Lumsdaine. Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks. In *Proceedings of the 2008 IEEE International Conference on Cluster Computing*. IEEE Computer Society, 10 2008.

[14] Parviz Kermani and Leonard Kleinrock. Virtual cut-through: a new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[15] Charles E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34(10):892–901, 1985.

[16] O. Lysne, T. Skeie, S. A. Reinemo, and I. Theiss. Layered routing in irregular networks. *Parallel and Distributed Systems, IEEE Transactions on*, 17(1):51–65, 2006.

[17] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. In *ACM Trans. on Modeling and Computer Simulations*, 1998.

[18] Li-Shiuan Peh and William J. Dally. A delay model and speculative architecture for pipelined routers. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, page 255, Washington, DC, USA, 2001. IEEE Computer Society.

[19] Harald Räcke. Minimizing congestion in general networks. In *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 43–52, Washington, DC, USA, 2002. IEEE Computer Society.

[20] Rachid Saad. Complexity of the forwarding index problem. *SIAM J. Discret. Math.*, 6(3):418–427, 1993.

[21] J. C. Sancho, A. Robles, and J. Duato. Effective strategy to compute forwarding tables for InfiniBand networks. In *Parallel Processing, International Conference on, 2001.*, pages 48–57, 2001.

[22] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, 9, 1991.

[23] Tor Skeie, Olav Lysne, and Ingebjørg Theiss. Layered Shortest Path (LASH) Routing in Irregular System Area Networks. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Washington, DC, USA, 2002. IEEE Computer Society.

[24] Herbert Sullivan and T R Brashkow. A large scale, homogeneous, fully distributed parallel machine, i. *SIGARCH Comput. Archit. News*, 5(7):105–117, 1977.

[25] The InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 1, Release 1.2*. InfiniBand Trade Association, 2003.

[26] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999.

[27] Brian Towles and William J. Dally. Worst-case traffic for oblivious routing functions. *IEEE Comput. Archit. Lett.*, 1(1), 2002.

[28] Brian Towles, William J. Dally, and Stephen Boyd. Throughput-centric routing algorithm design. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209, New York, NY, USA, 2003. ACM.

[29] Eitan Zahavi, Gregory Johnson, Darren J. Kerbyson, and Michael Lang. Optimized InfiniBand Fat-tree Routing for Shift All-To-All Communication Patterns. In *Proceedings of the International Supercomputing Conference 2007 (ISC07)*, Dresden, Germany.