Optimal Emulation of Meshes on Meshes of Trees

Alf-Christian Achilles*

Department of Computer Science
University of Karlsruhe
Germany
achilles@ira.uka.de

Abstract. Many problems can be solved more efficiently on a mesh of trees network than on a mesh. Until now it has been an open problem whether the mesh of trees is always at least as fast as the mesh. In this paper, we present an emulation of N-node meshes on $O(N)$-node meshes of trees with constant slowdown, even though any embedding of a mesh into a mesh of trees requires dilation $\Omega(\log N)$. This demonstrates that the mesh of trees is strictly more powerful than the mesh. As an application, we show how to construct an optimal $O(\sqrt{N})$ sorting algorithm for the mesh of trees that improves on the best previously known algorithm by a logarithmic factor.

1 Introduction

Fixed interconnection networks for parallel computers determine to a large degree the algorithms used. They define the available locality any algorithm must try to exploit to be efficient. One of the important questions regarding interconnection networks is their relative computational power which can be determined by emulating one network on the other. An emulation describes how any algorithm for one network can be transformed to run on another network with a defined slowdown. Efficient emulations thus contribute to the portability of algorithms across interconnection networks, sometimes even to the discovery of new, faster algorithms for certain problems, as is the case in this work regarding the problem of sorting.

In this work we show that the mesh of trees is strictly more powerful than the mesh by emulating the mesh on the mesh of trees with constant slowdown (open problem 2.5 in [6]), and use that result to show the existence of an optimal $\Theta(\sqrt{N})$ algorithm for the sorting problem on the mesh of trees.

In this paper we investigate the emulation of meshes by mesh of trees networks that are at most a constant factor larger than the emulated mesh. First we discuss the properties of the mesh of trees and present previous work on the emulation of meshes by meshes of trees. The following sections describe the

* This work was supported by the Graduiertenkolleg "Controllability of Complex Systems" (DFG Vo 287/5-2)

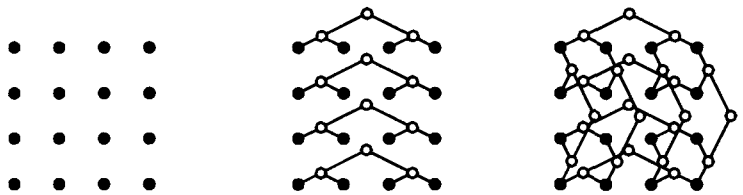

Fig. 1. A 4×4 mesh of trees: **a)** Mesh-like layout of the leaves. **b)** Addition of row trees. **c)** Addition of column trees.

emulation model used in this work and the application of neighborhood covers for the construction of the emulation. Finally we show how to parameterize the emulation such that both the expansion and the slowdown of the emulation are constant.

1.1 The Mesh of Trees

An $m \times m$ mesh of trees, $m = 2^i$ for $i \in \mathbb{N}$, is basically a 2-dimensional square mesh whose edges have been removed and have been replaced by complete binary trees on each column and row, thus additionally introducing new nodes in the internal nodes of the trees. Figure 1 shows a 4×4 mesh of trees.

An $M = m \times m$ mesh of trees contains $3m^2 - 2m = O(M)$ nodes and has a diameter of $4 \log m$. The mesh of trees is a recurrent network: By removing the root nodes and their incident edges in all trees the mesh of trees is partitioned into four disconnected $\frac{m}{2} \times \frac{m}{2}$ meshes of trees. Vice versa, a $2^i \times 2^i$ mesh of trees can be composed of 2^{2p}, $p \in \mathbb{N}$, $2^q \times 2^q$ submeshes of trees, $p + q = i$, by arranging the submeshes of trees in a grid pattern and adding the necessary tree nodes in each row and column.

The mesh of trees is an area-universal network, thus it can emulate any network of equal VSLI layout area with a polylogarithmic slowdown. Due to its small diameter, the mesh of trees can solve a number of problems more quickly than the 2-dimensional mesh. In particular, certain graph-theoretic problems such as the computation of the transitive hull or the minimum spanning tree can be solved on the N-node mesh of trees in time $O(\log^2 N)$, compared to $O(\sqrt{N})$ steps on the mesh. The capabilities of the mesh of trees are extensively discussed in [6].

However, there are problems whose best known solutions require asymptotically more steps on the mesh of trees than on the mesh. An important example is sorting N elements. The best previously known sorting algorithm for the mesh of trees needs $O(\sqrt{N} \log N)$ steps whereas on the mesh only $\Theta(\sqrt{N})$ steps are needed (see e.g. [12, 10, 5]).

1.2 Previous Work

Most of the work on interconnection networks has focused on finding efficient embeddings.

An embedding of a network $G = (V_G, E_G)$ into a network $H = (V_H, E_H)$ is a mapping of the nodes of G onto the nodes of H and a mapping of the edges of G onto paths in H such that the endpoints of an edge e are mapped to the endpoints of the image of e.

An embedding is generally characterized by three parameters: The *load* l of the embedding is the maximal number of nodes in G that are mapped to the same node in H. The *dilation* d is the maximal length of a path in H that is an image of an edge in G. The *congestion* c is the maximal number of paths in H that are images of an edge in G and share a common edge in H.

The emulation of G by H as described by an embedding consists of sending a packet along all the image paths in H for each step of G to be simulated and computing the next state for all the nodes of G. Both the dilation d and the congestion c are lower bounds for the transport of packets so that the transport takes $\Omega(d+c)$ steps. Since $\Omega(l)$ steps are needed for the computation of the next state of the nodes in G, at least $\Omega(d + c + l)$ steps are needed to emulate one step of the computation of G in H.

Previous work on the emulation of meshes by meshes of trees has relied on embeddings of the mesh into the mesh of trees. The intuitive embedding of an N-node mesh into an $O(N)$-node mesh of trees consists of a one-to-one mapping of the nodes of the mesh onto the leaf nodes of the mesh of trees and connecting the images of neighboring mesh nodes by paths in the trees. That embedding has load 1, congestion 2 and dilation $\Theta(\log N)$, which yields an emulation with a slowdown of $\Theta(\log N)$. In fact, this is also an asymptotically optimal embedding, since — as a consequence of the much more general result in [1] — any embedding of a mesh into a mesh of trees has dilation $\Omega(\log N)$. Therefore any emulation of meshes on meshes of trees that is based on embeddings must have slowdown $\Omega(\log N)$.

Probably due to the fact that the intuitive embedding already matches the dilation lower bound for embeddings, not much work has been done on this topic. A different model for emulations was needed that allows to overcome that dilation lower bound.

2 Emulations

There exists a more general but still realistic model for emulation of interconnection networks that has been proven to lead to stronger results.

A network is described by a directed graph $G = (V, E)$ with anti-parallel edges, where each vertex $v \in V$ in the graph designates a processor and each edge $e \in E$ a communication link between two processors. A computation on the network G is performed in disrete time steps as follows: at time step t each processor $v \in V$ has a state (v, t). To compute $(v, t + 1)$ it obtains a packet

(e,t) of constant length from one incoming communication link e incident to v, computes the next state $(v,t+1)$ as a function of this packet and (v,t) and generates a packet $(e',t+1)$ for one outgoing communication link e'.

An emulation of a network G on a network H begins with an initial distribution of all $(v,0)$, $v \in V_G$, to vertices in H. The emulation then proceeds by computing the states of the vertices in V_G for the next time steps according to the computation rules for G. At each emulation step, each processor u in H can receive an edge packet (e,t) from one of its incident incoming edges, compute the state of a node in G, if all the necessary data is present on u, and send a packet along one incident outgoing edge.

G has been emulated for T_G time steps if for every $v \in G$ there exists at least one (v,T_G) in one of the vertices of H. Let the number of steps in H needed for the emulation be T_H.

Definition 1. The *slowdown* S of an emulation of T_G steps of G in T_H steps on H is defined as $S = T_H/T_G$. The *expansion* $|V_H|/|V_G|$ indicates the relative size of G and H.

Only slightly different models have already been employed to construct universal parallel computers [8, 9], to emulate meshes on butterflies with constant slowdown [4], to show the equivalence of bounded-degree networks derived from the hypercube [11], to tolerate faults in some interconnection networks [7] and to emulate planar graphs on various classes of networks [2, 3].

Please note that the model makes no assumption about the size of the state of a processor (as in [4] and [2]), except that the computation of the next processor state takes constant time.

This emulation model allows to hide latency by pipelining the routing of information which has been computed locally with the aid of multiple copies of processor states to distant processors such that the amortized time necessary for the emulation decreases compared to emulations based on embeddings where the dilation is a lower bound for the slowdown.

3 Our approach

We use a more general emulation model than embeddings that allows processors of the host network to be emulated in more than one processors in the guest network, as described above. We will describe the use of neighborhood covers and show how to generate neighborhood covers for the mesh.

3.1 Emulation through neighborhood covers

The concept of t-neighborhood covers, formalized in [2], is very useful to construct emulations:

Definition 2. Let $G = (V,E)$ be a graph. The *t-neighborhood* of a vertex $v \in V$ consists of all the nodes in V whose distance to v is at most t. The t-neighborhood of a subset $V' \subset V$ is the union of the t-neighborhoods of all the nodes in V'.

A node $v \in V$ can be emulated for t steps by emulating its t-neighborhood, if the states of the nodes in the t-neighborhood at step $t = 0$ are known.

Definition 3. A *t-neighborhood-cover* of an undirected graph G is a set of subgraphs $\{S_1, S_2, \ldots, S_h\}$ of G such that for every v in G there is an S_i that contains the t-neighborhood of v in G.

Obviously, the graph G can be emulated for t steps by emulating t steps of the subgraphs in a t-neighborhood cover, since the t-neighborhood of each node in G is contained in some subgraph in the t-neighborhood. To proceed with the emulation beyond the first t steps information has to be exchanged between the subgraphs: Each node $v \in S_i$ with $(u, v) \in E$, $u \notin S_i$, obtains the edge packets $((u, v), 1), ((u, v), 2), \ldots, ((u, v), t)$ from the node $v \in S_j$ where S_j is a subgraph that contains the t-neighborhood of v and therefore has been emulated for the full t steps. $v \in S_i$ is called an *importer* and $v \in S_j$ the corresponding *exporter*.

The emulation of the subgraphs in a t-neighborhood cover can be performed recursively with the aid of neighborhood covers for the sugraphs. Finally, the subgraphs in the last level of the recursion are embedded into the emulating network. A node in the final subgraphs can be an exporter on each of the levels of the recursion and its emulating node in H must send edge packets to each of the nodes emulating a corresponding importer.

3.2 Generating neighborhood covers for the mesh

In the following we show how the neighborhood covers are recursively generated for the $n \times n$ mesh to be emulated. The levels of the recursion are denoted by k, $0 \leq k \leq \kappa(N)$, where $k = 0$ be the first and $\kappa(N)$ the last level.

Global neighborhood covers First we create global t_k-neighborhood covers for the complete $n \times n$-mesh by dividing the mesh into equally sized submeshes of decreasing size $n'_k \times n'_k$, which we call the *cores* of the neighborhood cover, such that they are aligned with the lower left corner (node $(0, 0)$) of the mesh (Figure 2a). The submeshes are then augmented by their $2t_k$-neighborhood (Figure 2b) to form a t_k-neighborhood cover consisting of $n_k \times n_k$ submeshes, where $n_k = n'_k + 4t_k$.

Actually, enlarging the submeshes by their t_k-neighborhood would have been sufficient to create a t_k-neighborhood cover, but the neighborhood cover we created has the convenient property that a corresponding exporter for each importer on the edge of an enlarged submesh is in the same column (row) in adjacent submeshes, such that any communication between submeshes needs to take place only vertically (horizontally). Figure 3 shows four overlapping submeshes and the position of the corresponding exporters and importers. A node that is an importer (exporter) along the horizontal direction or the vertical direction in a t_k-neighborhood cover will be called a *level k horiontal importer (exporter)* or a *level k vertical importer (exporter)*.

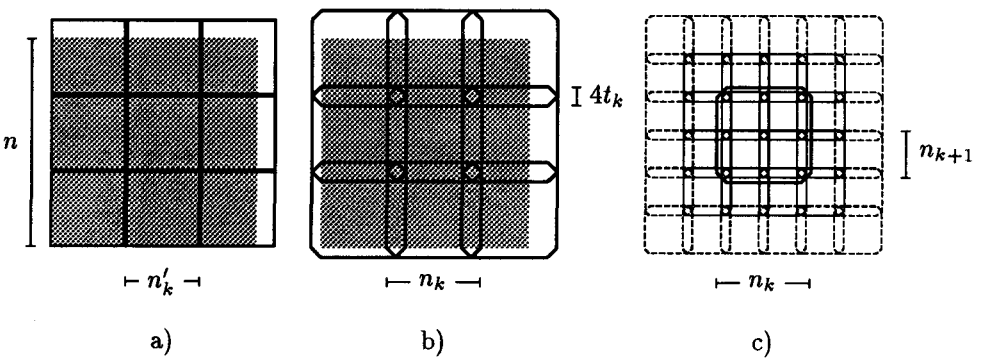

Fig. 2. Generation of a t_k-neighborhood cover in the mesh: **a)** Partitioning the mesh (grey area) into submeshes where some submeshes may extend beyond the borders of the mesh. **b)** Augmenting the submeshes by their $2t_k$-neighborhood. The border lines of the extended submeshes describe the set of exporters and importers. **c)** Recursive decomposition using those submeshes (solid lines) of the global t_{k+1}-neighborhood cover that have common nodes with a submesh in the t_k-neighborhood cover.

Bounding the number of importers and exporters in a row We will now take a close look at the set of nodes marked as exporters or importer by the global neighborhood covers up to level k. Each global t_k-neighborhood cover marks certain nodes of the mesh as exporters or importers. We will show that the nodes marked by all the neighborhood covers can be spread relatively evenly throughout the mesh. We require the following relationships between the t_k's and n'_k's, where $x \mid y$ denotes the fact that y is an integral multiple of x:

$$t_k \mid n'_k \tag{1}$$

$$t_{k+1} \mid t_k \tag{2}$$

$$t_k \geq n_{k+1} \tag{3}$$

Let P_k be a set of nodes in the mesh along equidistant parallel vertical, horizontal and diagonal lines of nodes in the $n \times n$ mesh:

$$P_k = \{(i,j) : (i \mid t_k) \vee (j \mid t_k) \vee ((i+j) \mid t_k) \vee ((i-j) \mid t_k)\}$$

It can easily be seen that with Equation 1 the exporters and importers in the t_k-neighborhood cover of the full mesh are contained in p_k (see Figure 2). Horizontal lines of nodes can contain vertical exporters and vertical lines can contain horizontal exporters, whereas diagonal can contain both types.

Furthermore, with the additional requirement of Equation 2 it follows that $P_{k+1} \subseteq P_k$ holds. Therefore all the nodes marked as exporters up to level k are contained in P_k.

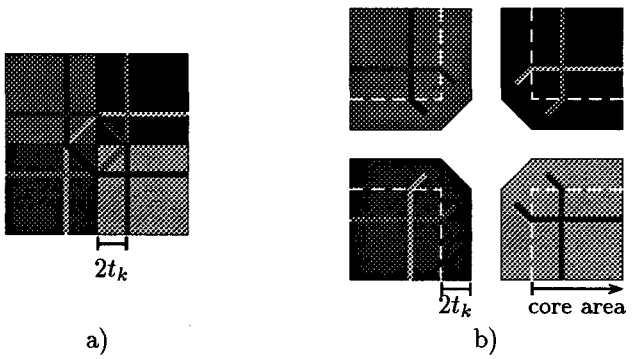

Fig. 3. Position of importers and exporters in adjacent submeshes in a t_k-neighborhood cover: **a)** Extension of the cores by their $2t_k$-neighborhood. The extension areas are delineated by lines of the same color as the core areas. These lines indicate the set of exporters of the neigborhood cover. **b)** Disentangled submeshes in the neighborhood cover showing the position of exporters (same color as the mesh they export to) corresponding to the importers (on the edge of adjacent submeshes). All exporters are at a distance of at least t_k from the borders of their respective submeshes.

Lemma 4. *At most* 3 *nodes per row (column) in any* $n_k \times n_k$ *submesh in the recursive neighborhood covers have been marked as horizontal (vertical) exporters or importers in levels* $< k$.

Proof. Since the parallel lines in P_k are t_k nodes apart we can infer that at most $\lceil n_{k+1}/t_k \rceil$ lines of each type traverse any $n_{k+1} \times n_{k+1}$ submesh. Both types of diagonal lines and the vertical (horizontal) lines can contain horizontal (vertical) exporters or importers and each line contributes one exporter or importer. With Equation 3 it follows that $3\lceil n_k/t_{k-1} \rceil \leq 3$. □

Generating recursive neighborhood covers Using the global t_k-neighborhood covers we recursively decompose the mesh into t_k-neighborhood covers: We start with the complete mesh which is decomposed using the global t_1-neighborhood cover. The decomposition proceeds recursively by decomposing each resulting $n_k \times n_k$ submesh into those $n_{k+1} \times n_{k+1}$ submeshes in the global t_{k+1}-neighborhood cover whose core areas share nodes with the submesh (Figure 2c). Conceptually, each submesh in a neighborhood cover will be emulated by repeatedly emulating the submeshes in the next stage.

4 Emulation of meshes on meshes of trees

To emulate an $N = n^2$-node mesh on an $m \times m$ mesh of trees with $O(m^2)$ nodes we recursively create t_k-neighborhood covers as described above until the

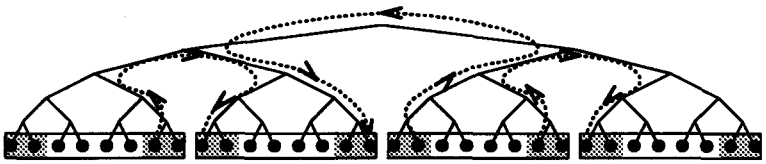

Fig. 4. Layout of paths connecting the exporters to the corresponding importers in one level of the emulation on the mesh of trees (*only some paths are shown*).

resulting submeshes are of a constant size $n_{\kappa(N)}$. These submeshes are embedded into the leaves of the smallest $m_{\kappa(N)} \times m_{\kappa(N)}$ mesh of trees that can contain them and these meshes of trees are recursively combined to $m_k \times m_k$ meshes of trees according to the recursive neighborhood covers to form the mesh of trees that emulates the complete mesh. Each $n_k \times n_k$ mesh will thus be emulated by an $m_k \times m_k$ mesh of trees.

4.1 Connecting importer and exporters

We need to route paths between the exporters and the importers on each level of the recursion within the submesh of trees that contains the neighborhood cover.

Note that for each level k horizontal (vertical) importer in the mesh of trees there is a level k horizontal (vertical) exporter on the same row (column) within the submesh of trees at level k that contains the importer. It follows that any communication paths between exporters and importers in the mesh of trees are contained either within only a column or within a row of the mesh of trees. For the rest of this paper we will therefore only investigate the communication between horizontal exporters and importers, since any observations can be applied to the communication between vertical row exporters and importers in an analogous and independent fashion. Hence, we will restrict ourselves to a description of the paths along the horizontal dimension of the mesh of trees as the other dimension is treated identically.

At each level k of the recursion we will assume that paths with constant congestion have already been laid out from each level $\leq k+1$ importer and exporter to the row roots of their respective $m_{k+1} \times m_{k+1}$ submeshes of trees.

We then need to connect the level $k+1$ importers and exporters by connecting the roots of adjacent meshes of trees as shown in Figure 4, generating a constant congestion.

Additionally we must provide paths from level $\leq k$ importers and exporters to the roots of the $m_k \times m_k$ mesh of trees from the roots of the respective submeshes of trees containing those exporters or importers. With Lemma 4 there are at most 3 exporters (importers) for the levels $< k$ and an additional 2 level k exporters and importers in in each row these paths only add a constant to the congestion.

Since the paths routed for each level of the emulation are on different levels of the mesh of tree the congestion is constant and the latency for any packet travelling along a path depends only on the length of the path which is bounded by twice the height of the tree ($2 \log m_{k-1}$).

4.2 Parameters for the recursive neighborhood covers

We now choose the parameters that determine how the t_k-neighborhood covers are generated at each level k of the recursion. All logarithms are to the base of two.

$$t_k = 2^{\lceil 2 \log \log n_{k-1} \rceil}$$

Since all t_k's are integral powers of two, Equation 2 holds. n'_k is the width of the cores generated at level k and n_k is the width of the extended submeshes at level k.

The number of submeshes of width n'_k in one row of a recursive neighborhood cover is at most n_{k-1}/n'_k. We set

$$\nu_k = 2^{\left\lfloor \log \frac{n_{k-1}}{\log^4 n_{k-1}} \right\rfloor} \qquad \text{and} \qquad n'_k = \left\lceil \frac{n_{k-1}}{\nu_k - 1} \right\rceil_{t_k}$$

where $\lceil x \rceil_p$ is defined as the smallest integral multiple of p that is less than or equal to x. It follows that ν_k is an upper bound on the number of submeshes in one row at level k. Furthermore ν_k is an integral power of two, allowing the combination of the submeshes of trees to form a complete, larger mesh of trees at each level of the recusion.

$$n_k = n'_k + 4t_k = \left\lceil \frac{n_{k-1}}{2^{\left\lfloor \log \frac{n_{k-1}}{\log^4 n_{k-1}} \right\rfloor} - 1} \right\rceil_{t_k} + 4 \cdot 2^{\lceil 2 \log \log n_{k-1} \rceil}$$

Note that the constraint required earlier in Equation 1 is satisfied. The recursion stops at level $\kappa(N)$ where $\kappa(N)$ is the first level such that that $n_{\kappa(N)}$ is smaller than some chosen constant or $t_{\kappa(N)-1} < n_{\kappa(N)}$ (thus complying with Equation 3).

It is easy to show that

$$\frac{\log^4 n_{k-1}}{2} \leq n_k = O(\log^4 n_{k-1}) \tag{4}$$

4.3 The size of the meshes of trees

In the following we show that the size of a mesh of tree at each level of the recursion is only by a constant factor larger than the mesh it is emulating. This information is needed to determine the lengths of the paths used to route packets between the exporters and the importers as well as to show that the complete

emulation has a constant expansion. The width m_k of a mesh of trees at level k is the number of submesh of trees multiplied by the width m_{k+1} of the submesh of trees:

$$m_k = \nu_{k+1} m_{k+1} \leq \frac{n_k}{n_{k+1}} \frac{n_{k+1}}{n_k} \nu_{k+1} m_{k+1}$$
$$\leq \frac{n_k}{n_{k+1}} \left(1 + \frac{1}{\frac{n_k}{\log^4 n_k} - 1} + \frac{20}{\log^2 n_k}\right) m_{k+1}$$

Therefore

$$m_k \leq \frac{n_k}{n_{\kappa(N)}} \prod_{i=k}^{\kappa(N)} \left(1 + \frac{1}{\frac{n_k}{\log^4 n_k} - 1} + \frac{20}{\log^2 n_k}\right)$$

With Equation 4, both $\sum_{i=0}^{\kappa(N)} \frac{1}{\frac{n_k}{\log^4 n_k} - 1}$ and $20 \sum_{i=0}^{\kappa(N)} \frac{1}{\log^2 n_k}$ are bounded by a constant for all N and the above product converges. Since $n_{\kappa(N)}$ is a constant we obtain

$$m_k = O(n_k)$$

Therefore the paths between exporters and importers at level k are $O(\log n_k)$ hops long and the overall emulation has constant expansion since $M = O(m_0^2) = O(n_k^2) = O(N)$.

4.4 Slowdown of the emulation

In this section we show that the emulation has constant slowdown. Let T_k be the number of steps needed to emulate a mesh of width n_k for t_k steps on an $m_k \times m_k$-MOT.

In each stage k of the recursive emulation, we emulate a mesh of width n_k for t_k steps by emulating the submeshes of width n_{k+1} for t_{k+1} steps and sending t_{k+1} packets from each exporter to its importer. This must be repeated $\left\lceil \frac{t_k}{t_{k+1}} \right\rceil$ times. The emulation of the submeshes takes T_{k+1} steps and, since the paths are of length $2 \log m_k = 2 \log O(n_k) = O(\log n_k)$, the last packet will arrive $O(\log n_k)$ steps later. Thus the time for the emulation of t_k steps on the $n_k \times n_k$ mesh is

$$T_k = \left\lceil \frac{t_k}{t_{k+1}} \right\rceil (T_{k+1} + O(\log n_k))$$
$$T_{\kappa(N)} = O(m_{\kappa(N)}) = O(n_{\kappa(N)} = O(1)$$

Note, that $\left\lceil \frac{t_k}{t_{k+1}} \right\rceil = \frac{t_k}{t_{k+1}}$ since $t_{k+1} \mid t_k$. The recurrence can be unfolded as

$$T_0 = \sum_{k=0}^{\kappa(N)-2} \left(\prod_{i=0}^{k} \frac{t_i}{t_{i+1}}\right) O(\log n_k) + T_{\kappa(N)-1} \prod_{k=0}^{\kappa(N)} \frac{t_k}{t_{k+1}}$$
$$= \sum_{k=0}^{\kappa(N)-2} \frac{t_0}{t_{k+1}} O(\log n_k) + O(1) \frac{t_0}{t_{\kappa(N)}} = t_0 \sum_{k=0}^{\kappa(N)-2} \frac{O(\log n_k)}{t_{k+1}} + O(t_0)$$

$$= t_0 \sum_{k=0}^{\kappa(N)-2} \frac{O(\log n_k)}{2^{\lceil 2 \log \log n_k \rceil}} + O(t_0) \leq t_0 O\left(\sum_{k=0}^{\kappa(N)-2} \frac{1}{\log n_k} \right) + O(t_0)$$

With Equation 4 we can infer that $\log n_{k+1} / \log n_k < 0.9$ for $n_k \geq 4$ and therefore the sume converges. Therefore we obtain $T_0 = O(t_0)$ and thus a constant slowdown:

$$S = \frac{T_0}{t_0} = O(1)$$

The result for the expansion and the slowdown of the emulation described above allows us to state the following.

Theorem 5. *Any N-node mesh can be emulated on an $O(N)$-node mesh of trees with constant slowdown.*

5 Sorting on the mesh of trees in optimal time

By emulating a sorting algorithm for the mesh we can also sort on the mesh of trees in the same asymptotic number of steps as on the mesh.

Lemma 6. *Any one-to-many offline routing problem on the leaves of an N-node mesh of trees can be solved in $O(\sqrt{N})$ steps.*

Proof. Omitted. □

Theorem 7. *N elements can be sorted on an $\sqrt{N} \times \sqrt{N}$ mesh of trees in time $O(\sqrt{N})$.*

Proof. Let the elements to be sorted reside in the leaves of an $\sqrt{N} \times \sqrt{N}$ mesh of trees. The leaves form a mesh which is emulated by an $O(\sqrt{N}) \times O(\sqrt{N})$ mesh of trees which is itself embedded into the $\sqrt{N} \times \sqrt{N}$ mesh of trees. To initialize the emulation we need to distribute the elements to be sorted to multiple leave nodes. This initialization of the emulation can be accomplished with a constant number of one-to-many offline routing phases, each taking $O(\sqrt{N})$ steps. We then use any of the well-known $O(\sqrt{N})$ sorting algorithm for the mesh and finally perform an inverse initialization to route the sorted elements to the correct leave node in the mesh of trees. □

6 Conclusion

We have shown that by recursively generating suitable neighborhood covers we can emulate a mesh on a mesh of trees of about the same size with constant slowdown, which is optimal except for a constant factor. This improves the slowdown of the previously best known emulation by a logarithmic factor.

This emulation is one of the rare cases where a general emulation actually also serves to discover a new, faster algorithm for an important problem: The emulation described above implicitly prescribes how to construct an algorithm for sorting on the mesh of trees in asymptotically optimal $O(\sqrt{N})$ steps.

The results presented here can easily be extended for higher dimensional meshes and meshes of trees.

References

1. S. N. Bhatt, F. R. K. Chung, J.-W. Hong, F. T. Leighton, and A. L Rosenberg. Optimal simulations by butterfly networks. In *Proceedings of the 20^{th} Annual ACM Symposium on Theory of Computing*, pages 192–204, May 1988.
2. Christos Kaklamanis, Danny Krizanc, and Satish Rao. New graph decompositions and fast emulations in hypercubes and butterflies. In *5th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 93)*, pages 325–334. ACM SIGACT, ACM SIGARCH, ACM Press, June 1993.
3. Christos Kaklamanis, Danny Krizanc, and Satish Rao. Universal emulations with sublogarithmic slowdown. In *Proceedings of the 34th IEEE Symposium Foundations of Computer Science (FOCS)*, pages 341–350, 1993.
4. Richard R. Koch, F. T. Leighton, Bruce Maggs, Satish B. Rao, and Arnold L. Rosenberg. Work-preserving emulations of fixed-connection networks. In *Proceedings of the 21^{st} Symposium on Theory of Computation*, pages 227–240, May 1989. Extended abstract.
5. M. Kunde. Routing and sorting on mesh-connected arrays. In J. Reif, editor, *Proceedings of the 3^{rd} Aegean Workshop on Computing: VLSI Algorithms and Architectures*, volume 319 of *Lecture Notes in Computer Science*, pages 423–433. Springer-Verlag, July 1988.
6. F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*, volume I. Morgan Kaufmann, San Mateo, CA 94403, 1992.
7. Tom Leighton, Bruce Maggs, and Ramesh Sitamaran. On the fault tolerance of some popular bounded-degree networks. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 542–552, October 1992.
8. Friedhelm Meyer auf der Heide. Efficient simulations among several models of parallel computers. *SIAM Journal on Computing*, 15(1):106–119, February 1986.
9. Friedhelm Meyer auf der Heide and Rolf Wanka. Time-optimal simulations of networks by universal parallel computers. In *Proceedings of the 6th STACS*, pages 120–131, 1989.
10. C. Schnorr and A. Shamir. An optimal sorting algorithm for mesh connected computers. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 255–263, May 1986.
11. Eric J. Schwabe. On the computational equivalence of hypercube-derived networks. In *2nd Annual ACM Symposium on Parallel Algorithms and Arichitectures*, pages 388–397. ACM, ACM Press, July 1990.
12. C. Thompson and H. Kung. Sorting on a mesh-connected parallel computer. *Communications of the ACM*, 20(4):263–271, 1977.