Dynamic Prioritization for Parallel Traversal of Irregularly Structured Spatio-Temporal Graphs

Bo Zhang
Duke University

Jingfang Huang
University of North Carolina at Chapel Hill

Nikos P. Pitsianis
Aristotle University

Xiaobai Sun
Duke University

Abstract

We introduce a dynamic prioritization scheme for parallel traversal of an irregularly structured spatio-temporal graph by multiple collaborative agents. We are concerned in particular with parallel execution of a sparse or fast algorithm for, but not limited to, an all-to-all transformation by multiple threads on a multi-core or many-core processor. Such fast algorithms play an escalating role as the basic modules to constitute, at increasingly large scale, computational solutions and simulations in scientific inquiries and engineering designs. We describe certain typical features of a spatio-temporal graph, not necessarily a tree, that describes a sparse or fast algorithm in execution. We show how the proposed dynamic prioritization scheme utilizes available but insufficient computing resources in practical and dynamic execution. We present experimental results with the application of this scheme to the celebrated fast multipole method.

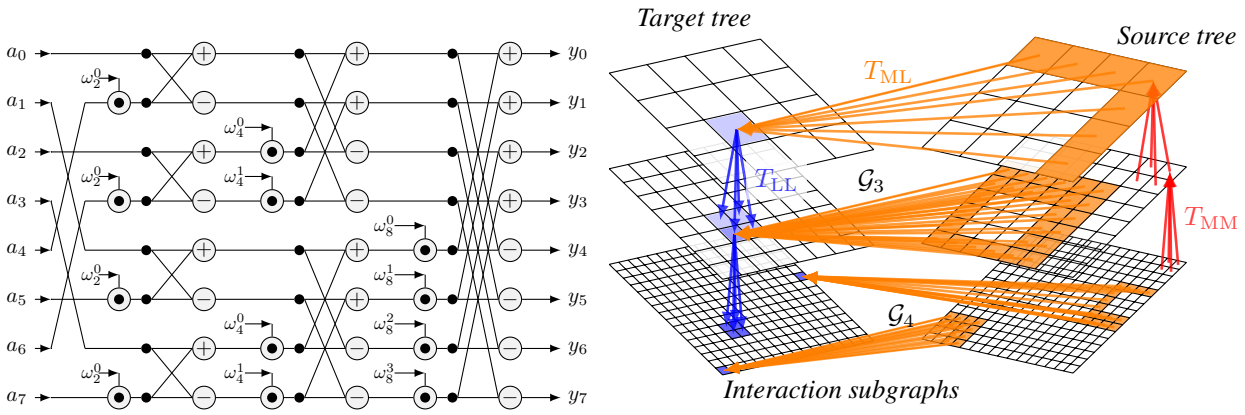
1 Description of ST-DAGs

The execution of a large-scale computation in the scientific or engineering studies [4, 5, 15, 17, 20] can often be analyzed and orchestrated as traversing a spatio-temporal directed graph. Typically, a node in the graph represents the computation associated with a particular spatial location, and a directed edge indicates the dependency from a predecessor node to a successor node. A spatial node at location $\mathbf{r}_i$ may be visited multiple times during the course of the computation, such as in an iterative or time-marching procedure. In analysis, we unfold such a spatial node into multiple spatio-temporal nodes $v_{ik} = (\mathbf{r}_i, k)$, where the temporal index k is local to the spatial index i, specifying the kth visit at the same location $\mathbf{r}_i$. This temporal unfolding process results in a spatio-temporal directed acyclic graph (ST-DAG). We may further assume that the computation at each node of a ST-DAG takes equal and hence one-unit time. A node requiring multi-unit time is expanded into multiple unit-time nodes. We refer to the nodes without predecessors or successors as the *frontier* or *terminal* nodes, respectively, and to the rest as the *interior* nodes.

The ST-DAGs we are concerned with include rooted trees but are not typically or necessarily trees. Specifically, while the out-degree of nodes in a tree equals to 1 uniformly, we consider the multiple out-degree at a non-terminal node v, $\deg^+(v) > 1$, as a typical characterization of many computation problems. A simple and common example is the pairwise interactions between a source ensemble, $\mathcal{S} = \{\mathbf{s}_j\}$, and a target ensemble, $\mathcal{T} = \{\mathbf{t}_i\}$. Such interactions may be represented by a matrix-vector product, or a system of linear equations, $\mathbf{y} = A\mathbf{x}$, where $\mathbf{y}_i$, for instance, is the potential at target $\mathbf{t}_i$ and $\mathbf{x}_j$ is the density at source $\mathbf{s}_j$. The interaction between $\mathbf{s}_j$ and $\mathbf{t}_i$ is specified by $A_{ij} = A(\mathbf{t}_i, \mathbf{s}_j)$. This computation can be represented by a bipartite graph $\mathcal{G}_{\rm B} = (\mathcal{S}, \mathcal{T}, \mathcal{E})$, where $(\mathbf{s}_j, \mathbf{t}_i) \in \mathcal{E}$ is a directed edge from $\mathbf{s}_j$ to $\mathbf{t}_i$ if $A_{ij} \neq 0$. The number of nonzero entries in the jth column of A is the out-degree of node $\mathbf{s}_j$, $\deg^+(\mathbf{s}_j)$. If $\deg^+(\mathbf{s}_j) = 1$, $\mathbf{s}_j$ interacts with only one target, which is not a typical situation.

The ST-DAGs of our interest are mostly evolved from an initially simple bipartite graph, representing algorithms with reduced arithmetic complexity in terms of the total number of nodes. The low arithmetic complexity is essential, and in strong demand, for enabling scientific computations or simulations at physically or biologically relevant scales. Such ST-DAGs encompass on the one hand sparse computations with introduced supernodes [12, 14] and on the other hand fast algorithms for dense computations based on various sparse factorizations or compressive representations. The latter, with increased and increasing applications, includes the fast Fourier transform (FFT) [6, 8], the Barnes-and-Hut (BH) algorithm [1, 2], and the fast multipole method (FMM) [3, 10, 11].

The fast algorithms, with provably minimal or sub-



(a) The FFT dependence graph.

(b) The uniform FMM graph.

Figure 1: Example of ST-DAGs for the FFT (left, taken from [7]) and the FMM (right).

minimal arithmetic complexity, leave parallelization as the last resort for further acceleration in computation, but impose new challenges as well. They have emerged with complex structures. We illustrate one simple graph for the FFT in Figure 1a, and another one for the FMM in Figure 1b. Each consists of multi-layered bipartite subgraphs. Neither is a tree. In graph representation, each can be seen as an aggregation of multiple sparse graphs stacked and stitched together. In the FMM, the input and output at an interior node are vector-valued. The FMM graph becomes highly irregular when it adapts to the spatial geometry and sampling distribution. This irregularity makes it hard to effectively employ parallelization schemes with regular or stationary patterns in space and time. And it prompts various structures to study.

We introduce a dynamic prioritization scheme for parallel traversing of a ST-DAG, permitting irregular structures, with multiple threads on a multi-core processor. We use the FMM as a non-trivial and comprehensive test of the general strategy (along with other parallelization schemes specific to the FMM). The dynamic prioritization scheme renders the minimal or sub-minimal traversal time with limited resources, not only in the static circumstance where the resource and computation time per node do not vary, but also in a dynamic circumstance as in a practical computation environment. We provide experimental results to show the effect of the strategy on parallel FMM performance.

2 The FMM graph

We attempt a graphical description of the FMM in its combinatorial aspect, pertinent to parallelization. In addition to its important role in scalable computation, the FMM is used here as a case study as well as a test of parallel schemes for its interesting, intrinsic, and comprehensive ST-DAG structures. The FMM graph may be better described by its subgraphs and the operations they represent.

2.1 The FMM tree

The FMM tree represents a hierarchical partition of a bounded spatial domain. The root node at the top level, or level 0, represents the smallest bounding box in $\mathcal{R}^d$, $d = 1, 2, 3$, that contains N source and target spatial samples. A box at level $\ell \geq 0$ is further partitioned equally along each dimension if it contains more than a prescribed number of points. Otherwise, the node associated with the box becomes a leaf node. Nodes corresponding to empty sub-boxes are pruned from the tree. With the node-box correspondence, we use node(s) and box(es) interchangeably. Let $\ell_{\max}$ be the index to the bottom level. When source and target clusters are uniformly distributed in the initial bounding box, the tree is likely full and well-balanced, yielding $\ell_{\max} = O(\log N)$.

Strictly speaking, the FMM tree is a fusion of two spatial partitions in one spatial hierarchy. One partitions the source ensemble, and the other partitions the target ensemble, see Fig. 1b for a separated view of the source and target trees in red and blue colors, respectively.

2.2 Intra-scale bipartite traversal

The FMM represents compressively the pairwise interactions at multiple scales. We describe the bipartite subgraph $\mathcal{G}_\ell$ at each level $\ell \in [2, \ell_{\max}]$. The vertex set of $\mathcal{G}_\ell$ includes $\mathcal{V}_\ell^s$ and $\mathcal{V}_\ell^t$, the respective sets of source and target boxes in the FMM tree at level ℓ, and the edge set represents the source-target interactions. A source box $\mathcal{B}_s$

and a target box $\mathcal{B}_t$ at level ℓ are connected by an edge if (1) $\mathcal{B}_s \cap \mathcal{B}_t = \emptyset$, and (2) their parents are near-neighbors, i.e., with common boundary (see Fig. 1b). The bipartite interaction subgraphs $\mathcal{G}_\ell$ at different levels are disjoint from each other. When the source and target clusters are uniformly distributed, $\mathcal{G}_{\ell+1}$ has 2^d times as many nodes as $\mathcal{G}_\ell$ does. Often, the trees are far from well-balanced, as a result of the adaptation to the domain geometry and sample distribution. Each of the bipartite interaction subgraphs is subsequently irregular. In the FMM nomenclature, the traversal operations on the scale-level subgraphs are referred to as the multipole-to-local translations, denoted by T_{ML} (see Fig. 1b).

The source tree, target tree, interaction graphs, and their inter-connections constitute the FMM graph.

2.3 Inter-scale tree traversal

Besides its precise analysis on accurate computation, the FMM surpasses also in arithmetic complexity the so-called tree codes (such as the BH algorithm) by introducing inter-scale translations. The FMM computation traverses the source tree, the interaction graphs, and the target tree in the following partial ordering. (1) Upward traversal on the source tree, up to level 2. Every leaf node in the source tree computes and renders a vector-valued output data, namely, the multipole expansion coefficients. A non-leaf node accumulates and translates data from its child nodes into its own multipole coefficients. Such upward inter-scale operations are known as the multipole-to-multipole translations, denoted by T_{MM} (see Fig. 1b). (2) Bipartite traversal of the interaction subgraphs $\mathcal{G}_\ell$, namely, the multipole-to-local translations T_{ML}, as described in Section 2.2. (3) Downward traversal of the target tree. At level $\ell > 2$, every node gets the local coefficients at its parent, merges them with its own at level ℓ, and makes the integrated results available to its child nodes or residing target locations. These downward inter-scale operations are known as the local-to-local translations, denoted by T_{LL} (see Fig. 1b). By their spatial relationships, the levelwise interaction subgraphs are overlaid at the tree nodes in the data structures of many FMM implementations.

At any non-leaf node of the FMM tree, the traversal operations follow the same dependency order: upward, horizontal and downward. A naive way to satisfy the dependency constraints at all the spatial nodes is to carry out the computation in three synchronized stages. It is, however, customary in the sequential FMM implementation that the horizontal operations are interleaved with the downward operations. A parallelization from such a sequential ordering would be very limited in performance by the obscured concurrencies. Although the concurrency among the upward or downward tree traversal operations are well understood and exploited, the multiple-way concurrency across the upward (T_{MM}), horizontal (T_{ML}) and downward (T_{LL}) traversal operations had been waiting for long to be exposed and explored. The dynamic prioritization approach we describe in the next section allows us to exploit efficiently such multi-way concurrency with various and irregular spatio-temporal structures.

3 Dynamic prioritization

The objective of parallel traversing a ST-DAG is to reach the shortest time possible under certain constraints. We shall discuss first on the minimal-time parallel traversing in an ideal, constraint-free situation where we have sufficient resources to employ at our disposal and we are assisted by a mechanism that invokes the computation at a node the instant its input data become available. We then introduce a dynamic prioritization scheme for parallel traversal subject to a resource constraint and dynamic variation in computation time among the nodes.

3.1 Absolute ranking

The shortest time in the constraint-free case can be achieved by following a simple data-driven rule. The frontier nodes start computation immediately and unconditionally, which are removed together with their incident edges upon the completion of their computations. In the reduced ST-DAG, nodes with available input data become the new frontier nodes and start their computation, and so on. The latest time step among all the terminal nodes in the original ST-DAG is the absolute shortest time. The predecessors of the final-step terminal nodes are time critical. Within the same period, however, it is not necessary to start every non-critical node at the earliest possible time. In the constrained situation, as we shall show shortly, we utilize the slack period at any non-critical node between the time its input becomes available and the latest time the computation must start. To this end, we define the absolute node ranking as follows. Adopting Hu's methodology [13], we introduce an artificial sink node to the graph under consideration as the sole successor to all the original terminal nodes. The absolute rank of a node v is the length of the longest path connecting v and the sink node. With this ranking system, one can invoke the nodes of the highest ranks among the frontier nodes at any given time while maintaining the shortest time in parallel traversal.

We define also the unconditionally sufficient number of processing elements (PEs) as the maximal number of nodes in concurrent computation at any time, across all possible schedules for parallel traversing in the absolutely shortest time.

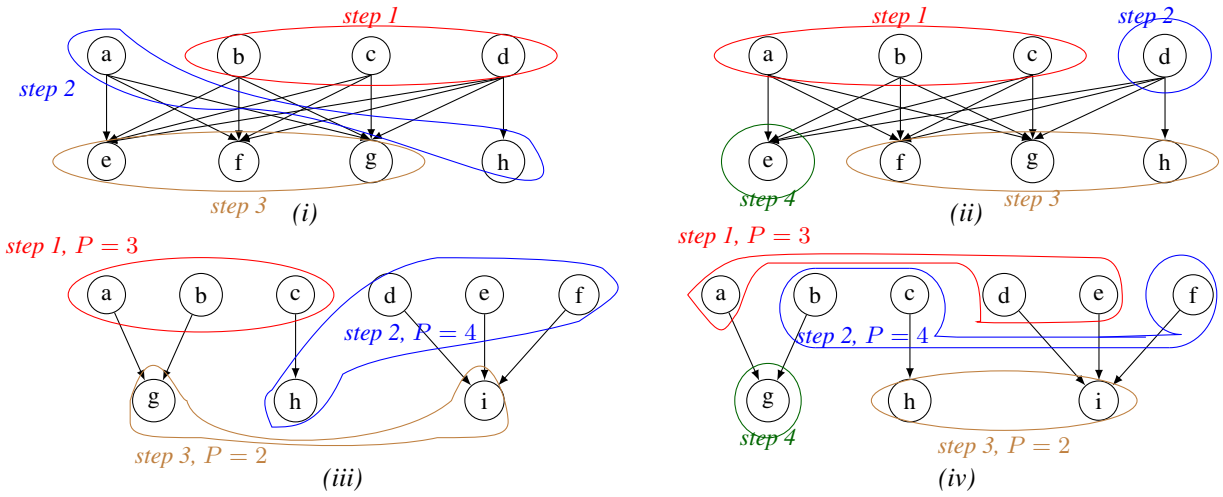


Figure 2: Parallel traversing with (left) or without (right) the dynamic prioritization on a non-tree DAG with $P = 3$ (top), on a tree with $P = 3, 4, 2$ (bottom), respectively.

3.2 Conditional ranking and prioritization

Parallel computation in a practical environment is subject to limited resources, such as the number of PEs and variation in latency due to memory accesses, thread switches, time sharing, and etc. The constrained minimization in traversal time is further complicated on highly irregular graphs. We introduce in this section the concept of adaptive ranking among the frontier nodes.

Assume first the resource constraint. The number of available PEs, P, is far smaller than sufficient for the absolutely shortest time case. At any step, however, when there are no more than P frontier nodes, the frontier nodes are served all at once, just as in the constraint-free situation. Otherwise, the nodes of higher ranks are served first.

Consider the case there are two frontier nodes u and v of the same rank, but only one of them can be served together with the other $P-1$ nodes of higher ranks. A random tiebreaker is a candidate policy for utilizing the parallel capacity at the current step. However, it is not an optimal policy for any DAG towards the shortest time under resource constraint. It is necessary to look ahead and maximize the number of frontier nodes at the next steps, which influence the subsequent steps. We want to maintain maximal degree of concurrency at any and all steps. We are also concerned with the execution dynamics. The set of the frontier nodes deviate substantially due to the dynamics which we shall discuss shortly in Section 3.3. We associate each frontier node c with an ordered pair, $\langle d_m^-(c), n_s(c)\rangle$, where

$$d_m^-(c) = \min_{(c,d)\in\mathcal{E}(\mathcal{G})} \deg^-(d). \tag{1}$$

When $d_m^-(c) = 1$, $n_s(c)$ equals to the number of direct successors of c with in-degree 1. That is, c is the only node in the way to release and enable these successor nodes. Otherwise, $n_s(c)$ equals to the out-degree of c, $\deg^+(c)$. The completion of c makes its successors one step closer to the front. Back to the selection between equal-ranked nodes u and v for service by one available PE. We give u a higher priority if

$$\langle d_m^-(u), -n_s(u)\rangle < \langle d_m^-(v), -n_s(v)\rangle \tag{2}$$

in the lexicographical order. A random selection breaks the tie in (2). Suppose u is chosen. Upon u's completion, each of its direct successors decrements its in-degree by one. The ranking criterion (2) may seem counterintuitive in the case $d_m^-(u) = 1$, as opposed to the case $d_m^-(u) > 1$. They are consistent in that we deal in the former case with immediate node release in order to maximize the concurrency at the very next step, and in the latter case, with the greatest reduction in parallel dependency for maximizing the concurrency in the future steps.

We illustrate in Fig. 2 the distinction between the dynamic prioritization scheme, to the left, and that by static ranking and random tiebreaker, to the right. The remaining portion of the same DAG in the final two traversal stages (in the constraint-free sense) is shown, without drawing the sink node explicitly. Nodes a, b, c, and d are of the same absolute rank. With $P = 3$, only three nodes can be served at the current step. Two schemes are compared. The scheduling on the left follows the adaptive ranking (2), see the three-step assignments enclosed in the red, blue, and brown curves with marked orderings. When the first assignment uses a random selection instead, oblivious to its impact on the concurrency degree in future steps, the same work takes four steps to

complete, as shown on the right, in the red, blue, brown and green curves. Consider the case that the DAG is a tree. When the number of PEs remains constant, the priority scheme by the adaptive ranking (2) is as good as the random tiebreaker. When the number of PEs varies in the course of traversal, however, the adaptability in (2) shows its advantage, see the comparison at the bottom of Fig. 2, where the available PEs increase from 3 to 4 and then drop to 2. Detailed analysis will be provided elsewhere. We remark that the problem of parallel traversing a DAG of N nodes in the shortest time, with the number of PEs constant or varying, is in general NP-complete, see [18].

3.3 Adaptability to traversal dynamics

The ranking and prioritization scheme described above is adaptable, in a straightforward way, to practical situations with multiple sources of dynamics. Among others, the computation at the nodes may be asynchronized or non-systolic, the latency in memory accesses is not uniformly distributed, the input to a high-ranked node may not be made available prior to that of a low-ranked node, and the number of available PEs may fluctuate during the course of traversal. We use the Pthreads library [16] in implementation. The scheme is friendly to thread dynamics in creation, switching and termination. It is executed by all participating threads in collaboration, using priority queues, without reserving one for central control. We also leverage the release-relay property in the thread management. Once created, a thread moves, upon completing the computation at the current node, to another frontier node of top priority, as long as the set of frontier nodes remains non-empty.

4 Experimental results and discussion

The experiments were carried out on a workstation with two 12-core AMD 6168 processors and 64GB memory. We used the system function *pthread_set_affinity_np* to select and activate a partial or the entire set of the cores for evaluating scalability. For the numerical task, we computed the pairwise interactions of N particles governed by the Yukawa potential $e^{-\lambda \mathbf{r}}/\mathbf{r}$ using the FMM, where $\mathbf{r}$ is the Euclidean distance between two interacting particles and λ is a modest positive constant. Particles were randomly distributed within a normalized unit cube in $\mathcal{R}^3$. The top bar chart in Fig. 3 shows parallel efficiency in the weak scaling aspect, namely, the ratio between the problem size N and the number of cores P remains constant. The ordinate axis is the parallel efficiency. We used as the base line the sequential code [19], one of the best in practical use. We present two cases. The ratio is 3.75E6 for 3-digit accuracy, shown in blue bars, and 6.25E5 for 6-digit accuracy, shown in red bars. The parallel efficiency remains above 90% in each case, in the presence of thread overhead. Without the dynamic priority scheduling, the parallel efficiency was above 90% up to 8 cores but dropped to 60% with 24 cores.

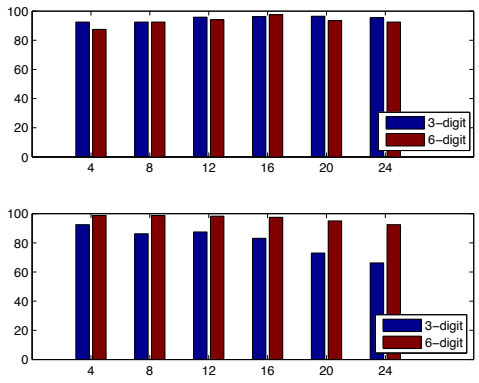


Figure 3: Parallel efficiency in weak (top chart) and strong (bottom chart) scaling aspects.

The bottom bar chart shows the parallel efficiency in the strong scaling aspect, with the number of cores increasing from 4 to 24, while keeping the problem size N at 1.5E7. We show two cases. The first, shown in blue bars, attains 3-digit accuracy. The second case, shown in red bars, reaching 6-digit accuracy, maintains above 90% parallel efficiency, as it involves more numerical operations at each and every computation node in response to a higher accuracy requirement. The first case lacks computation work to keep all 24 cores busy most of the time. This is in part caused by the operation aggregations to counterbalance the overhead in thread management. The balance between the granularity in concurrent operations and the overhead in thread management is delicate.

This work on parallel FMM is preceded by many related works over the last two decades, from the first and insightful one by Greengard and Gropp [9] to the most recent and spectacular implementation and application by Rahimian *et al* [17]. The exploitation of multiway concurrency in the FMM and the application of the dynamic prioritization in this work seem new and lead to significantly improved parallel performance on multicore processors.

The authors thank the referees for their critical and constructive comments on the initial manuscript.

References

[1] APPEL, A. W. An Efficient Program for Many-Body Simulation. *SIAM Journal on Scientific and Statistical Computing 6* (1985), 85–103.

[2] BARNES, J., AND HUT, P. A Hierarchical $O(N \log N)$ Force-Calculation Algorithm. *Nature 324* (1986), 446–449.

[3] CARRIER, J., GREENGARD, L., AND ROKHLIN, V. A Fast Adaptive Multipole Algorithm for Particle Simulations. *SIAM Journal on Scientific and Statistical Computing 9* (1988), 669–686.

[4] CHEW, W. C., JIN, J. M., MICHIELSSEN, E., AND SONG, J. M. *Fast and Efficient Algorithm in Computational Electromagnetics*. Artech House, 2001.

[5] CHEW, W. C., TONG, M. S., AND HU, B. *Integral Equations Methods for Electromagnetic and Elastic Waves*. Morgan & Claypool, 2008.

[6] COOLEY, W., AND TUKEY, J. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation 19* (1965), 297–301.

[7] CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. *Introduction to Algorithms*. The MIT Press, 2001.

[8] FRIGO, M., AND JOHNSON, S. G. The Design and Implementation of FFTW3. *Proceedings of the IEEE 93*, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[9] GREENGARD, L., AND GROPP, W. A Parallel Version of the Fast Multipole Method. *Computers & Mathematics with Applications 20* (1990), 63–71.

[10] GREENGARD, L., AND ROKHLIN, V. A Fast Algorithm for Particle Simulations. *Journal of Computational Physics 73* (1987), 325–348.

[11] GREENGARD, L., AND ROKHLIN, V. A New Version of the Fast Multipole Method for the Laplace Equation in Three Dimensions. *Acta Numerica 6* (1997), 229–269.

[12] HÉNON, P., RAMET, P., AND ROMAN, J. On Finding Approximate Supernodes for an Efficient Block-ILU(k) Factorization. *Parallel Computing 34* (2008), 345–362.

[13] HU, T. C. Parallel Sequencing and Assembly Line Problems. *Operations Research 9* (1961), 841–848.

[14] LIU, J. W. H., NG, E., AND PEYTON, B. W. On Finding Supernodes for Sparse Matrix Computations. *SIAM Journal on Matrix Analysis and Applications 14* (1993), 242–252.

[15] LU, B., CHENG, X., HUANG, J., AND MCCAMMON, J. Order N Algorithm for Computation of Electrostatic Interactions in Biomolecular Systems. In *Proceedings of the National Academy of Sciences* (2006), vol. 103, pp. 19314–19319.

[16] MUELLER, F. A Library Implementation of POSIX Threads under UNIX. In *Proceedings of the USENIX Conference* (1993), pp. 29–41.

[17] RAHIMIAN, A., LASHUK, I., VEERAPANENI, S. K., CHANDRAMOWLISHWARAN, A., MALHOTRA, D., MOON, L., SAMPATH, R., SHRINGARPURE, A., VETTER, J., VUDUC, R., ZORIN, D., AND BIROS, G. Petascale Direct Numerical Simulation of Blood Flow on 200K Cores and Heterogeneous Architectures. In *SC 10': Proceedings of the 2010 ACM/IEEE Conference on Supercomputing* (2010).

[18] ULLMAN, J. D. NP-Complete Scheduling Problems. *Journal of Computer and System Sciences 10* (1975), 384–393.

[19] ZHANG, B., HUANG, J., PITSIANIS, N. P., AND SUN, X. Revision of FMM-Yukawa: An Adaptive Fast Multipole Method for Screened Coulomb Interactions. *Computer Physics Communications 181* (2010), 2206–2207.

[20] ZHAO, D., HUANG, J., AND XIANG, Y. A New Version Fast Multipole Method for Evaluating the Stress Field of Dislocation Ensembles. *Modeling and Simulation in Materials Science and Engineering 18* (2010).