Hierarchical Parallelization of Molecular Fragment Analysis on Multicore Cluster

Liu Peng[1], Bhupesh Bansal[1], Ashish Sharma[2],
Rajiv K. Kalia[1], Aiichiro Nakano[1], Priya Vashishta[1]
[1]Collaboratory for Advanced Computing and Simulations
University of Southern California, Los Angeles, CA, 90089, USA
Email: {liupeng, rkalia, anakano, priyav}@usc.edu, bbansal.usc@gmail.com
[2]Center for Comprehensive Informatics
Emory University, Atlanta, GA 30322, USA
Email: Ashish.Sharma@emory.edu

Abstract—*Molecular fragment analysis, using connected component identification algorithm, is of great significance for structural and chemical analysis in computer aided material design. However, it is a great challenge to accelerate molecular fragment analysis due to the scale, diversity and irregularity of molecular graphs. To address this challenge, we propose a hierarchical parallelization approach consisting of: (1) inter-node parallelization via spatial decomposition and hook-and-contract algorithm; (2) inter-core parallelization via master-and-worker scheme; and (3) locality optimization based on space-filling curve to improve memory accessing. Experiments show that the proposed scheme achieves nearly linear inter-node strong scalability up to* 50 *million vertices molecular graph on 32 computing nodes, and over 13-fold inter-core speedup on 16 cores. The experiments also demonstrate the effectiveness of locality optimization on performance enhancement.*

1. Introduction

Massive data analysis on parallel computers has become an essential part, and often a bottleneck, of scientific computing. For example, large-scale molecular dynamics (MD) simulation, which has become an integral part of computer aided material design, often involves multimillion atoms and computing the molecular fragments (or connected components) is essential for the structural and chemical analysis, such as identifying molecular products during the combustion of fuels [1]. Challenges of molecular fragment analysis mainly arise from three aspects: (1) complexity introduced by scale of molecular graph–high-end MD simulations typically involve multimillion atoms [2], which amounts to multimillion vertex-sized graph, imposing great difficulty for molecular fragment identification, and therefore it is important to design efficient parallel algorithm to harvest computing power; (2) diversity introduced by molecular graph density and connectivity–due to the wide range of MD simulations, for example, simulation sizes can range from $100,000$ atoms (10MB of data per frame) to 1 billion atoms (100GB), the resulting molecular graph can vary from dense to sparse, from a small number of large fragments to a large number of small fragments. Fig. 1 illustrates the variation in data characteristics starting from a large number of small molecular fragments at the beginning of the simulation to a nearly connected single fragment in the last snapshot with the increased percolation level; (3) irregularity introduced by MD datasets–MD datasets are inherently irregular, so are the molecular graph generated from them, and serial access to the molecular graph datasets often exhibits poor spatial and temporal locality, which leads to ineffective use of memory hierarchy [3]. Therefore, it is a great challenge to accelerate molecular fragment analysis.

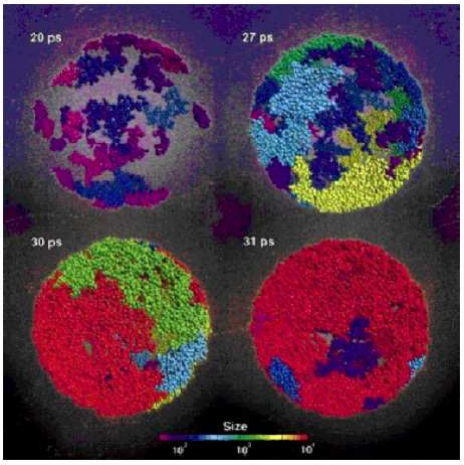

Fig. 1: Snapshot of a molecular dynamics simulation to study the percolation of OAl_4 cluster during the oxidation of an aluminum nanocluster, where color represents the size of molecule fragments.

Problem Statement: Let $G = (V, E)$ represent an undirected

molecular graph, where the vertex set V consists of atoms and the edge set E contains bonds between pairs of atoms. Let i and j represent two atoms; then edge $(i, j) \in E$ *iff* there is a bond between atom i and atom j. The problem is to identify all connected components in the molecular graph. We refer to vertex as an atom in the graph.

There are some special features in the molecular graph:

- *Vertex degree restriction*: The vertex degree has an upper limit. For example, in the simulation of SiO_2, the vertex degree, for each silicon atom, is typically four and, for each oxygen atom, is on average two. Due to the nature of the chemical bonding, even outliers do not exceed the maximum node degree, typically set around 10. And this restricts the number of expanding vertices in the breadth first search (BFS) algorithm.
- *Boundary restriction*: Due to the active interatomic interaction cutoff R_c, scientists are usually interested in only atoms within the cutoff distance. This restricts the depth in the depth first search (DFS) algorithm.

The major contributions of this paper is a hierarchical parallelization approach on a multicore cluster, combining:

- Spatial decomposition and hook-and-contract algorithm are utilized effectively for inter-node parallelization.
- Master-and-worker scheme is used for inter-core parallelization.
- Locality optimization via space-filling curve is employed to improve memory accessing.

The rest of this paper is organized as follows. After summarizing related works in section 2, section 3 presents the hierarchical parallelization scheme, including hierarchical decomposition, inter-node and inter-core parallelization. And section 4 details our locality optimization via space-filling curve. Section 5 presents the experimental results and analysis. Finally, section 6 concludes the paper.

2. Related Work

There have been great efforts on the study of connected component identification. Generally, there are three approaches : (1) Graph traversal, such as DFS and BFS based approach; (2) Graph adjacency matrix transitive closure based approach; (3) Hook-and-contract based approach, where vertices are hooked together to form a large set of vertices and then outgoing duplicate edges and internal edges to the set are removed to contract the set to a single super-vertex, and the process is repeated until the maximally connected components are found. Hirschberg, Chandra, and Sarwate [4] presented an $O(\log^2 n)$ algorithm using $n^2/\log n$ processors on the CREW PRAM model, where n is the number of vertices. Chin [5] later reduced the processor requirement to $n^2/\log^2 n$. Johnson and Metaxas [6] proposed a CREW algorithm with $O(\log^{1.5} n)$ complexity using $n+m$ processors, where m is the number of edges. Chong and Lam [7] presented an $O(\log n \log\log n)$ algorithm using $n + m$ processors for the EREW model. However, these fine-grained parallel algorithms require impractical number of computing nodes for molecular fragment analysis which usually involves millions of atoms. Consequently, researchers have also studied coarse-grained parallel algorithms for connected component analysis. Chin [5] designed an $O((m \log n)/p + \log n)$ algorithm for up to $p = n/\log n^2$ processors, in PRAM model, for dense graphs. Kruskal [8] presented faster parallel algorithms for sparse graph with $O(m/p + (n \log p)/p + p^{1+\epsilon})$ complexity for the EREW model and $O(m/p + (n \log p)/p + p \log p)$ for the CREW model, where ϵ is an negligible value. All these approaches assume shared memory, which is not the case for cluster–the current most popular platform.

3. Hierarchical Parallelization

In this section, we present our hierarchical hook-and-contract algorithm. In the following subsections, we detail inter-node parallelization and inter-core parallelization, respectively.

3.1 Inter-node Parallelization

In this subsection, we first describe how to decompose the molecular graph and assign decomposed subgraphs to computing nodes. Then we describe a hook-and-contract algorithm for inter-node parallelization implemented via Message Passing Interface (MPI).

- *Inter-node decomposition.* 3D mesh decomposition is employed to divide the whole molecular graph into smaller subgraphs, and each subgraph is mapped to a processor in an array of $P_x \times P_y \times P_z$ computing nodes, where P_x, P_y, and P_z are even positive integers. The purpose of decomposition is to assign equal load to each computing node, thereby achieving load balancing, which is known to be an effective way to improve the parallelization efficiency of irregular applications [9]. Specifically, our scheme partitions the whole molecular graph (in terms of atoms) in a computational space, which is related to the physical space by a curvilinear coordinate transformation: The computational space shrinks where the workload density is high and expands where the density is low, so that the workload is uniformly distributed. To minimize the load imbalance and communication costs as a function of the coordinate transformation, our approach employs simulated annealing to figure out the optimal range information. The range information of each computing node is then propagated to its 26 neighbors in a 3D cube.
- *Inter-node hook-and-contract algorithm.* After the spatial decomposition, each computing node is assigned a chuck of atomic data, i.e. a list of a subset of atoms. Then, in step 7 of Alg. 1, each node performs independent $Singlenode\_Fragment\_Analysis$ using a graph traversal algorithm shown in Alg. 2. This

Algorithm 1: Inter-node parallel algorithm for molecular fragment analysis. n is the number of atoms and p is the number of computing nodes.

input : 1. an atom bond file containing adjacency lists of atoms.
2. an atom dataset file containing atom attributes[1].

output: Fragment list

1: Each node gets a chunk of input files.
2: **for** node 1 to p in parallel **do**
3: Load balancing to distribute atoms at each node;
4: Each node gets x, y, z range information for its 26 direct neighbors;
5: **end for**
6: **for** Node 1 to p in parallel **do**
7: Run $Singlenode\_Fragment\_Analysis$;
8: **end for**
9: **for** i from 1 to $\log_2 p$ **do**
10: **for** j from 0 to $(p-1)/2^i$ in parallel **do**
11: Node pair $(1 + j \times 2^i,\ 1 + j \times 2^i + 2^{i-1})$ hook together: (Fragments, adjacency lists and cross-atom lists are merged into node $1 + j \times 2^i$.)
12: **end for**
13: **end for**
14: Return fragments from node 1.

[1] Atom dataset file is used to filter atomic dataset based on some simulation attribute if needed

Algorithm 2: Inter-core parallel algorithm for molecular fragment analysis.

input : Atom list A. $A[i].nbr$ is atom $A[i]$'s adjacency list, $A[i].threadId$ is initialized to -1. t is the number of threads.

output: Local fragment list FL

Master thread:

1: Shuffle the atom list A.
2: **while** not all atoms in A marked **do**
3: Pick the first t unmarked atoms in A and store them in $root$.
4: Create a new graph G' with t vertices and no edges.
5: $\forall 1 \leq i \leq t$, create fragment $frag_i$.
6: Start t worker threads.
7: Wait for all worker threads to finish.
8: Run BFS to identify connected components in G'.
9: Merge fragments inside one connected component in G'.
10: Append distinct $frag_i$ to FL.
11: **end while**

Worker thread i:

1: do a depth-limited search starting from $root[i]$:
2: **for** each atom a visited during search **do**
3: Put a in $frag_i$ and mark a's fragment id as i.
4: If a's fragment id is already marked as j, create an edge between vertex $v[i]$ and $v[j]$ in G')
5: **end for**

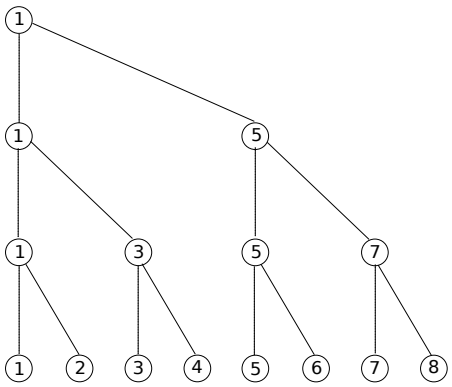

Fig. 2: Inter-node merge example for 8 nodes.

analysis produces three outcomes: (1) it tags each atom with appropriate fragment identification number; (2) it saves local fragments; and (3) it saves the list of cross-atoms connected to each fragment, where a cross-atom is defined as follows: For vertex i, j, if $(i, j) \in E$ then j is a cross-atom of i *iff* i and j belong to different nodes. This step is done in $O(n'/q + q)$ time where n' and q are the number of atoms and number of cores per computing node respectively.

Steps 9-12 in Alg. 1 work as a hook-and-contract mechanism, which consists of $\log_2 p$ iterations. At the ith iteration, $(p-1)/2^i$ group of pair-wise nodes does information merge in parallel. Specifically, nodes $1 + j \times 2^i$ and node $1 + j \times 2^i + 2^{i-1}$, for $j = 0$ to $(p-1)/2^i$, merge their fragment lists, adjacency lists and cross-atom lists to node $1 + j \times 2^i + 2^{i-1}$. Merging fragment list may merge two fragments, if one fragment contains any cross-atom of atoms from the other fragment, into one fragment. After $\log_2 p$ iterations, all merges are done and we get global graph-level cross-node fragments. Fig. 2 illustrates this hook-and-contract scheme. Each tree node represents a computing node in cluster. During iteration 1, group of nodes $(1, 2)$ merge their fragment lists, adjacency lists and cross-atom lists to node 1. Groups $(3, 4)$, $(5, 6)$ and $(7, 8)$ do the same merge to node 3, 5, 7 respectively. After this iteration, nodes $1, 3, 5, 7$ together carry all fragments information. During iteration 2, groups $(1, 3)$ and $(5, 7)$ do the same merge again with results in node 1 and 5 respectively. Finally during iteration 3, group $(1, 5)$ merges to node 1, which carries the final fragment information.

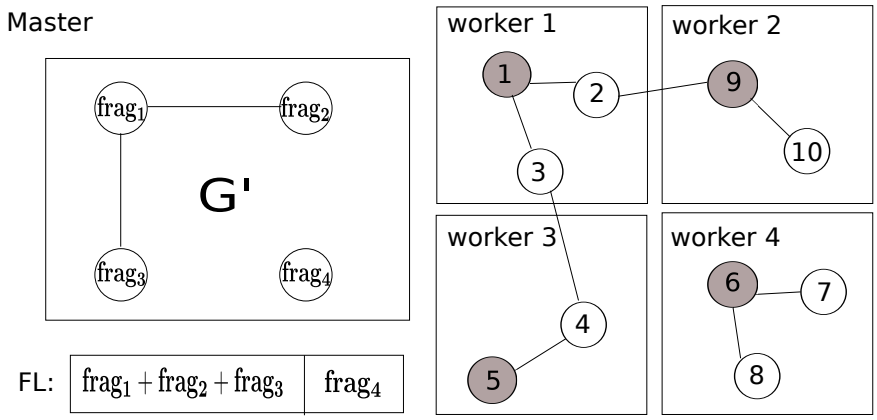

Fig. 3: Master/worker inter-core example.

3.2 Inter-Core Parallelization

Alg. 2 shows the inter-core parallelization for fragment analysis. A master/worker model is employed to accelerate local fragment analysis: Each worker thread independently detects a local fragment through graph traversal algorithm and then the master thread is in charge of merging fragments from each worker thread.

Specifically, the master thread shuffles the whole atom list at step 1. The purpose is to make the t atoms (t is the number of workers), to be used as DFS root by each worker thread, selected at step 3 randomized. At step 4, the master thread creates a new graph G' with vertex i representing the fragment $frag_i$ to be generated by worker thread i. Next, t worker threads are started and they work in parallel to detect $frag_i$ and create edges in G'. After all worker threads are finished, the master thread runs BFS to detect connected components in G'. At step 9, any two fragments belonging to the same connected component in G' are merged. Final fragments are added to local fragment list at step 10.

Each worker thread does a depth-limited DFS based graph traversal to take advantage of material characteristics, e.g. interatomic interaction cutoff distance. At step 3, each thread i keeps marking unvisited atoms as part of $frag_i$. If it encounters an atom already marked in $frag_j$, then it creates an edge between vertex i and j in G' as described at step 4.

Fig. 3 illustrates inter-core parallel algorithm involving one master and four workers. The left part describes the graph G' created by the master, where each vertex $frag_i$ represents the fragment generated by worker i and the edges are created by workers. On the right side, each vertex represents an atom and DFS root atoms are shaded. Each rectangle shows a worker's working region. Workers $1, 2, 3, 4$ run depth-limited DFS, in parallel, starting from roots $1, 9, 5, 6$, respectively, keeping expanding their working regions independently. When one worker i expands to an atom already visited by another worker j, then an edge is created in G' for $frag_i$ and $frag_j$. For example, when worker 1 tries to expand from atom 2 to atom 9 which is already visited by worker 2, an edge between $frag_1$ and $frag_2$ is added to graph G'. After all workers finish, the master traverses G', merge $frag_1, frag_2, frag_3$ into one fragment, denoted as $frag_1 + frag_2 + frag_3$, and append all fragments to FL. The above procedures repeat until all atoms are visited.

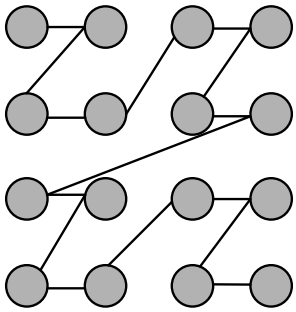

Fig. 4: Z Curve to sort the input data.

4. Locality Optimization

MD datasets are irregular, and serial access to them often exhibits poor spatial and temporal locality, which leads to ineffective use of a memory hierarchy [3]. It has been shown that data reordering (i.e., strategy to change data management to increase data reuse in memory) can significantly improve memory hierarchy utilization [3]. In MD simulations, a space-filling curve (e.g., Morton curve or Z curve illustrated in Fig. 4), is often used to preserve spatial locality [10]. A space-filling curve is a mapping of a one-dimensional array to three-dimensional grid points, which preserves the spatial proximity of successive array elements [3]. Our optimization strategy organizes input atomistic simulation data according to a space-filling curve. We implemented Z curve to improve the data locality and further to enhance the performance of molecular fragment analysis.

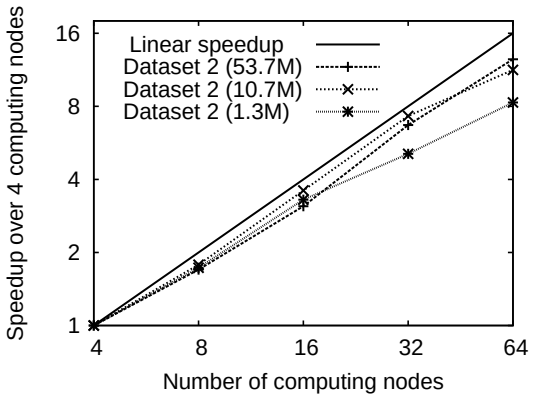

Fig. 5: Inter-node strong-scaling speedup comparison with different sizes of RDX crystal datasets2 up to 64 computing nodes.

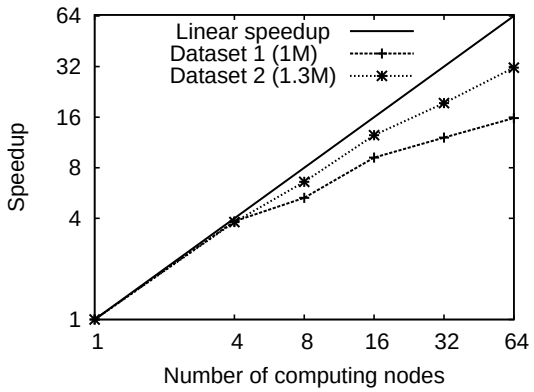

Fig. 6: Inter-node strong-scaling speedup comparison for 1 million-atom shocked RDX dataset1 and 1.3 million-atom RDX crystal dataset2 up to 64 computing nodes.

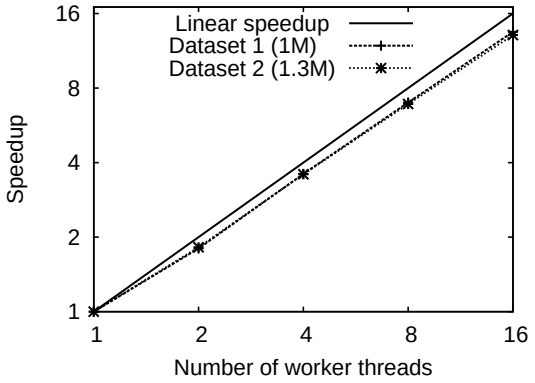

Fig. 7: Inter-core strong-scaling speedup comparison for 1 million-atom shocked RDX dataset1 and 1.3 million-atom RDX crystal dataset2 up to 16 worker threads.

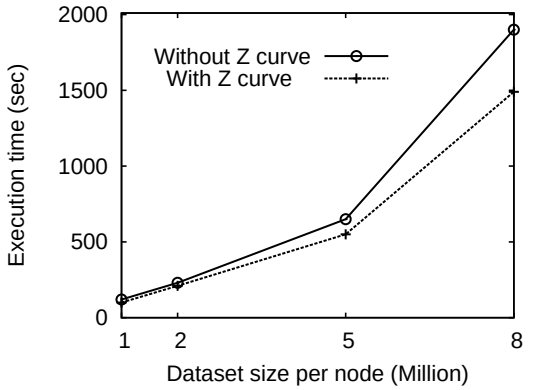

Fig. 8: Execution time as a function of dataset size per node in million atoms with and without Z curve technique.

5. Performance Evaluation

To test the effectiveness of the proposed optimization methods, we conduct three sets of experiment testing the performance of inter-node parallelization, inter-core parallelization and locality optimization.

First, we test the performance of inter-node parallelization. Strong scalability is used as a metric for fixed-size problem. We have conducted scalability tests with two datasets, each of different sizes–approximately 1 million atoms, 10 million atoms and 50 million atoms–on a Linux cluster consisting of 133 Intel Xeon dual-processor (2.8GHz) nodes with 2GB memory per node and connected by Myrinet interconnect. Dataset 1 is an atomistic dataset for shock-compressed RDX (1,3,5-trinitro-1,3,5-triazine) crystal with a large number of long chains and highly dense molecules, whereas dataset 2 represents a normal-density RDX crystalline structure with regular molecules having small chains of atoms and moderate density.

Fig. 5 compares the strong-scaling speedup over 4 computing nodes for different sizes–1.3, 10.7 and 53.7 million atoms–of the RDX crystal dataset (dataset 2). The speedup is nearly linear up to 32 computing nodes for 10.7 and 53.7 million atoms, while it decreases to less than 32 for 64 computing nodes. The reason is that as the number of computing nodes increases, the hierarchical merge takes more time, which degrades the performance. Fig. 6 compares the speedup of two datasets of a similar size but different characteristics (1 million-atom shocked RDX data, dataset 1, and 1.3 million-atom RDX crystal, dataset 2). Dataset 2 exhibits better speedup, which is probably due to the smaller number of cross-atoms.

Next, we test the inter-core strong scalability. We use the same datasets as the inter-node testing in Fig. 6, and we use a 16-core SMP platform consisting of 4 core i7 quadcore processors (Nehalem 920). Fig. 7 shows that our inter-core parallelization exhibits excellent speedup up to 16 worker

threads with over 13 speedup for both datasets. Namely, the speedup is material characteristic independent.

Finally, we test the effectiveness of our locality optimization approach. Fig. 8 compares the performance with and without Z curve based locality optimization for dataset 1 of different sizes: 1, 2, 5, 8 million atoms, respectively, on a single core i7 processor with 12GB memory. The figure demonstrates the effectiveness of space-filling Z curve in increasing data locality, which improves the performance up to 21%.

6. Conclusion

This paper studied molecular fragment analysis using hierarchical parallelization to harvest computing power of multicore cluster. We have combined three approaches: (1) inter-node parallelization via spatial decomposition and hook-and-contract algorithm; (2) inter-core parallelization via master-and-worker algorithm; (3) locality optimization based on space-filling curve to improve memory accessing. Experiments showed that our proposed scheme achieves almost linear inter-node strong scalability up to 50 million-atom sized molecular graph up to 32 computing nodes, and over 13 inter-core speedup on 16 cores. Also experiments demonstrate the effectiveness of locality optimization on performance enhancement. However, the inter-node performance degrades when the number of computing nodes exceeds 64, which suggest the need for a better inter-node merging algorithm.

7. Acknowledgement

This work was supported by NSF-PetaApps/EMT, DOE-SciDAC/SciDAC-e/BES/EFRC, and DTRA.

References

[1] A. Strachan, A. C. T. van Duin, D. Chakraborty, S. Dasgupta, and W. A. Goddard, "Shock waves in high-energy materials: The initial chemical events in nitramine rdx," *Phys. Rev. Lett.*, vol. 91, p. 098301, 2003.

[2] A. Nakano, R. K. Kalia, K. ichi Nomura, A. Sharma, P. Vashishta, F. Shimojo, A. C. T. van Duin, W. A. Goddard, R. Biswas, D. Srivastava, and L. H. Yang, "De novo ultrascale atomistic simulations on high-end parallel supercomputers." *Int'l J. High Performance Comput. Appl.*, vol. 22, pp. 113–128, 2008.

[3] J. Mellor-Crummey, D. Whalley, and K. Kennedy, "Improving memory hierarchy performance for irregular applications using data and computation reorderings," *Int'l J. Parallel Program.*, vol. 29, pp. 217–247, 2001.

[4] D. S. Hirschberg, A. K. Chandra, and D. V. Sarwate, "Computing connected components on parallel computers," *Commun. ACM*, vol. 22, pp. 461–464, 1979.

[5] F. Y. Chin, J. Lam, and I.-N. Chen, "Efficient parallel algorithms for some graph problems," *Commun. ACM*, vol. 25, pp. 659–665, 1982.

[6] D. B. Johnson and P. Metaxas, "Connected components in $o(\log^{3/2} v)$ parallel time for the crew pram (extended abstract)," in *Proceedings of the 32nd annual symposium on Foundations of computer science*, ser. SFCS '91. Washington, DC, USA: IEEE Computer Society, 1991, pp. 688–697.

[7] K. W. Chong and T. W. Lam, "Finding connected components in $o(\log n \log \log n)$ time on the erew pram," *Journal of Algorithms*, vol. 18, pp. 378–402, 1995.

[8] C. Kruskal, L. Rudolph, and M. Snir, "A complexity theory of efficient parallel algorithms," in *Automata, Languages and Programming*, ser. Lecture Notes in Computer Science. Springer Berlin, 1988, vol. 317, pp. 333–346.

[9] A. Nakano, "Multiresolution load balancing in curved space: the wavelet representation." *Concurrency: Practice and Experience*, vol. 11, pp. 343–353, 1999.

[10] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the hilbert space-filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, pp. 124–141, 2001.