Distributed Algorithm of Dynamic Multidimensional Data Mapping on Multidimensional Multicomputer in the LuNA Fragmented Programming System

V.E.Malyshkin[1,2,3] and G.A.Schukin[1,3]

[1] Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of Russian Academy of Sciences, Novosibirsk, Russia
[2] Novosibirsk National Research University, Novosibirsk, Russia
[3] Novosibirsk State Technical University, Novosibirsk, Russia
{malysh, schukin}@ssd.sscc.ru

Abstract. The distributed algorithm Patch with local communications for dynamic data allocation of a distributed multicomputer in the course of an application LuNA fragmented program execution is presented. The objective of the Patch is to decrease the length and as result the volume of communications while the application parallel program is executed. Communications include all the internode interactions for data processing, dynamic data allocation, search and balancing. The Patch takes into account the data dependencies and maximally tries to keep the data locality during all the internode interactions.

Keywords: Dynamic Data Allocation, Dynamic Load Balancing, Distributed Algorithms with Local Interactions, Fragmented Programming Technology.

1 Introduction

Supercomputer large-scale numerical modeling is now widely used, especially in science. To achieve good performance and scalability of application parallel programs of numerical modeling the effective resources allocation strategies, control, dynamic load balancing and other means usually should be used. Because of that, the complexity of application parallel programming becomes comparable to the complexity of system parallel programming. To simplify the development of parallel programs of numerical modeling, the LuNA system for automatic construction of parallel programs was developed ([1-6]). We consider that in visible future LuNA-like systems should significantly or fully eliminate parallel programming from the process of large-scale numerical models creation.

The LuNA automatically assembles an application parallel numerical program out of pieces (fragments) as data as computations. Each fragment of computations (CF) in the course of a LuNA-program execution defines an independent process. Each CF computes the output data fragments (DF) values from CF's input DFs values. Each DF is the single assignment variable of a LuNA-program and each CF is executed once only. Fragmented structure of a LuNA-program is kept during the LuNA-program execution. This allows to provide DFs and CFs migration between the nodes of a multicomputer and their execution in parallel.

Efficiency of the LuNA fragmented program execution substantially depends on the quality of distributed resources allocation. Below a distributed Patch algorithm with local interactions is described. Patch is intended for dynamic distributed allocation of distributed resources. It is included into the LuNA system and optimized for allocation of multidimensional data on grid-like communication network.

2 Related Works

There are many domain decomposition methods for data distribution and load balancing. One of them is tiled arrays ([7-10]), where data are decomposed into orthogonal convex tiles, which are then distributed over nodes of a multicomputer. In [7] and [9] hierarchical tiling arrays are used. In [10] arbitrary tiles (i.e. the tiles can have different sizes by any dimension and not necessarily be aligned) are allowed. In [8] user-defined decomposition of arrays into tiles is supported. Restriction of tiled arrays is rectangular shape of tiles, which in some cases can prevent achievement of load balance between tiles. Also, decomposition on tiles is assumed to be static and little information is given about possibility of its dynamic change in a distributed way.

Data decomposition into domains is also widely used in molecular dynamics and particle simulations ([11-15]). In [11] and [13] two or three dimensional domain (field of particles) is partitioned by placing inner vertices into it, thus creating a mesh of domains, each is initially of rectangular form; one domain is assigned to each PE of a multicomputer. For a purpose of load balancing inner vertices can be moved to change domains size and PEs' workload. Load balancing can proceed in distributed form, i.e. each node communicates only with its neighbors, but sometimes global operations are used. Usually convex shape of domains is required, which may sometimes be a restriction. Also up to 8 PEs (in 3D case) can share a single inner vertex, so a movement of this vertex can require a synchronization between all these PEs.

In [14] recursive orthogonal bisection is used for domain decomposition. For load balancing planes of bisections are recursively moved to change domains size. Because nodes and domains assigned to them are organized in a tree, load balancing requires tree traversal and consequently communications between unfixed number of non-neighboring nodes. Also tree topology may not fit for an actual grid network topology.

In [12] domains consist of Voronoi cells which can be moved between domains to achieve load balance. In [15] usual rectangular cells are used in a similar fashion. To preserve communication pattern between neighboring nodes, some cells are marked as permanent, i.e. can't be moved. Because these algorithms are for molecular dynamics applications and use additional information (such as particles and their movement) for load balancing decisions, they may not be readily applicable for more general cases.

We can conclude that desired data distribution and load balancing algorithm should be distributed, use preferably local communications, and be applicable to a broad range of problems.

3 Distributed Algorithm of Dynamic Data Allocation

Previously, a distributed Rope algorithm for dynamic data allocation was developed [1-2]. Rope algorithm was designed to support general data structures processing on distributed network. This paper presents new algorithm Patch, which is developed to support mesh data structures processing on distributed grid network, where Rope algorithm has some disadvantages. Further sections present description of Patch algorithm and its comparison with Rope.

3.1 Initial Definitions

Numerical algorithms data are usually multidimensional Cartesian meshes (for example, particles-in-cell or Poisson equation solver meshes). For data decomposition the mesh is divided into parallelepipeds (DFs) that form the grid of DFs. Two DFs, whose planes are adjacent, are called adjacent DFs. Two DFs are called neighbor DFs if the value of one of them is computed by application algorithm with the use of the other DF value.

3.2 Patch Algorithm

Rope algorithm used mapping of multi-dimensional mesh of DFs to one-dimensional numerical domain (for example, with Hilbert space-filling curve) for data distribution [16]. To better exploit multidimensional neighborhood, multi-dimensional numerical domain is required. For this reason, the Patch algorithm maps k-dimensional mesh of DFs to n-dimensional numerical domain. The numerical domain is represented as Cartesian grid of regular cells, each cell has its n-dimensional coordinate. Mapping to n-dimensional domain allows all neighbor DFs be mapped to the same or adjacent cells, thus, neighborhood relation on DFs is better preserved. Mapping is fixed during program execution.

Assuming n-dimensional Cartesian grid network topology is used, the whole numerical domain is decomposed into subdomains (“patches”) of cells, one subdomain for each node (Fig. 1, left). Cell coordinates are globally ordered through the nodes by all dimensions. There are no restrictions on a shape of domains, but several constraints should be satisfied:

- No empty subdomains are allowed
- Square of subdomain adjacent planes should be minimal in order to minimize the volume of communications between adjacent subdomains
- Each node's subdomain should be adjacent to subdomain of it's neighbor nodes in a network topology

These constraints are to enable proper functioning of data allocation algorithm, which is described further.

In LuNA system each DF and each CF are mapped dynamically on the nodes of distributed multicomputer at runtime. Additionally, migration of CFs and DFs also demands dynamically to search where CFs and DFs are located. Thus, data allocation algorithm should provide assignment of any DF on any node and search for any DF from any node. Node, which currently holds a cell to which a DF was mapped, is called a residence of this DF. If DF's residence is known, the DF can be allocated on the residence node or requested (copied) from it.

To make possible a DF residence determination from any node, in Patch algorithm each cell stores information about current location (node) of all its adjacent cells in a Cartesian grid. The use of this adjacency information makes it possible to find DF's residence from any node for a finite number of steps, using only local communications. First, cell coordinate for required DF is acquired. This coordinate is a constant and can be computed everywhere because DF mapping to cells is fixed. Secondly, if current node doesn't contain a cell with required coordinate, the cell, closest to it in the current node's "patch", is chosen and search can be continued from the neighbor node, adjacent to it in the direction to the required cell. Otherwise, search is over and the residence is found.

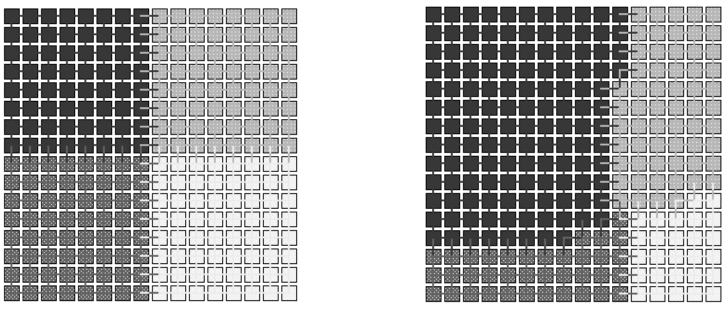
Fig. 1. Distribution of cells on PEs in Patch algorithm, initial (left) and after load balancing (right). Different colors denote different PEs. Adjacency links are shown also. See [16] for dynamic picture.

Benefits of Patch algorithm are follows:

- Preserves neighborhood of DFs in all dimensions
- Ideally fits for an actual grid network topology
- The worst case DF allocation time is proportional to the diameter of the network topology, which is usually smaller then Rope's algorithm (total number of nodes)

3.3 Dynamic Load Balancing in Patch Algorithm

Diffusion scheme is used in Patch algorithm for dynamic load balancing implementation. All nodes of a multicomputer in a grid topology are divided into overlapping groups of nodes. Each group consists of a central node and all its neighbor nodes in the topology of communicating system. Each cell, assigned to a node, defines a node workload, which is calculated by some formula or criteria, for example, a value of a current total volume of the DFs, on the node mapped to the cell. Total load of a node is a sum of the loads of all cells on the node. Every node keeps the value of its total load and the values of total loads of all its neighbor nodes. Node is considered overloaded, if its total load is greater than average load of all cells of the group, and underloaded otherwise. If a central node of a group is overloaded/underloaded, then some cells should migrate to/from underloaded/overloaded nodes of the group equalizing their workload (Fig. 1, right). All the CFs, that process migrated DFs, follow the DFs.

Migrated cells are selected in such a way in order to minimize the number of border cells between subdomains, because this number is directly proportional to the volume of communications between the neighbor DFs. Usually, this means selecting connected group of cells lying on subdomains border. Greedy algorithm is used for cells selection. Migration is actually implemented, if the difference between total and average workloads exceeds a threshold.

To synchronize simultaneous exchange of cells between many nodes and keep cells information consistent, the following transaction mechanism is used. Each transaction comprises transfer of cells from one node to another. If cells are transferred between two nodes, then any other transactions between them are locked. To prevent deadlocks and decide which transaction should be executed next, a random priority is assigned to each transaction. Transaction with max priority is executed first. Because a cell can be adjacent to many nodes' subdomains (not actually participating in transaction), special updates are sent to these nodes to keep adjacency information correct.

Transferring single cells instead of shifting a whole subdomain border allows for more accurate load balancing. Also only two nodes need to be synchronized for each transaction, which allows for more transactions to be done in parallel. Only local communications between neighbor nodes are used during load balancing, which makes the algorithm scalable.

4 Tests

To compare the algorithms, LuNA implementation of Poisson equation solver with an explicit finite-difference scheme on a regular 3D mesh was chosen. Experiments were conducted on cluster with two Intel Xeon E5-2690 processors per PE and Infiniband FDR communication and transport network. GCC 5.3 C++ compiler and MPICH 3.2 MPI library were used.

4.1 Test Results

For testing the regular mesh of 512^3 size was divided into 32^2 three dimensional data fragments. Measurements were done for up to 256 processes (N). Cluster configuration allowed placement up to 4 processes on a single physical node. 1D and 2D grid network topologies were used for testing Rope and Patch algorithms respectively.

Following program execution characteristics were measured: total execution time (ET, seconds), average DF send distance (AvgSD, processes in topology), average summary size of all DFs sent by process (AvgSS, megabytes; also includes DFs movement during load balancing), standard deviation from an average computation time per process (AvgCTD, seconds).

Table 1 shows the results when all processes are loaded uniformly and load balancing doesn't need. Showing comparable results of execution time and average migrated data size, Patch, as expected, demonstrates much better average DFs migration distance than Rope due to exploiting of multidimensional DF neighborhood.

For dynamic load balancing testing initial disbalance was created assigning data and computations on one half of processes only. The goal of balancing was to load all processes as equally as possible. Table 2 shows results of the testing. Patch algorithm generally demonstrates smaller deviation of average computational time, i.e. it managed to load processes better than Rope. Also average DFs migration distance and total size of transferred data are smaller for Patch again because of exploiting more dimensions than Rope. Increased average transferred data sizes and decreased migration distances, comparing with uniform distribution case, are due to data movement during load balancing.

Table 1. 512^3 mesh, 32^2 fragments, uniform distribution.

N	1	2	4	8	16	32	64	128	256
ET, Rope	773.8	407.4	217.8	111.9	57.9	31.7	19.7	14.7	17.4
ET, Patch	769.0	409.4	221.3	113.4	68.8	31.7	19.4	12.3	13.6
AvgSD, Rope	0	1	1.5	1.8	2.5	3.24	4.84	6.41	9.66
AvgSD, Patch	0	1	1	1	1	1	1	1	1
AvgSS, Rope	0	20.2	20.2	20.2	15.1	12.6	8.8	6.9	4.7
AvgSS, Patch	0	20.2	20.2	20.2	15.1	12.6	8.8	6.9	4.7

Table 2. 512^3 mesh, 32^2 fragments, non-uniform distribution with dynamic load balancing.

N	2	4	8	16	32	64	128	256
AvgCTD, Rope	53.4	15.99	55.68	47.21	25.18	13.51	7.44	3.93
AvgCTD, Patch	58.11	23.7	56.49	33.33	21.39	12.56	7.01	3.71
AvgSD, Rope	1	1.19	1.42	1.48	1.46	1.80	1.30	1.44
AvgSD, Patch	1	1.03	1.18	1.27	1.31	1.24	1.16	1.04
AvgSS, Rope	7787.6	6246.7	2821.6	1032.2	1192.6	598.0	566.9	392.0
AvgSS, Patch	9917.9	4955.9	2225.3	1226.2	714.6	444.1	193.8	91.8

5 Conclusion

The problematics of data allocation and load balancing automation for implementation of large-scale numerical models for supercomputers are considered. The Patch algorithms for dynamic multidimensional data allocation in LuNA fragmented programming system is proposed. Comparison tests of the algorithm are presented.

References

1. Malyshkin, V.E., Perepelkin, V.A., Schukin, G.A.: Scalable distributed data allocation in LuNA fragmented programming system. J. Supercomputing, vol. 73, no. 2, pp 726-732. Springer US (2017).
2. Malyshkin, V.E., Perepelkin, V.A., Schukin, G.A.: Distributed Algorithm of Data Allocation in the Fragmented Programming System LuNA. In: PaCT 2015, LNCS, vol. 9251, pp. 80-85. Springer, Heidelberg (2015).
3. Malyshkin, V.E., Perepelkin, V.A.: LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. In: PaCT 2011, LNCS, vol. 6873, pp. 53-61. Springer, Heidelberg (2011).
4. Malyshkin, V.E., Perepelkin, V.A.: Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System. J. Supercomputing, vol. 61, no. 1, pp. 235-248 (2012).
5. Malyshkin, V.E., Perepelkin, V.A.: The PIC Implementation in LuNA System of Fragmented Programming. J. Supercomputing, vol. 69, no. 1, pp. 89-97 (2014).
6. Kraeva, M.A., Malyshkin, V.E.: Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. J. Future Generation Computer Systems, vol. 17, no. 6, pp. 755-765 (2001).
7. Gonzalez-Escribano, A., Torres, Y., Fresno, J., Llanos, D.R.: An Extensible System for Multilevel Automatic Data Partition and Mapping. J. IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 5, pp. 1145-1154. IEEE (2014).
8. Chamberlain, B.L., Deitz, S.J., Iten, D., Choi, S.-E.: User-Defined Distributions and Layouts in Chapel: Philosophy and Framework. In: HotPar'10, 2nd USENIX conference on Hot topics in parallelism, pp. 12-12. USENIX Association, Berkeley, CA, USA (2010).
9. Bikshandi, G., Guo, J., Hoeflinger, D., Almasi, G., Fraguela, B.B., Garzarán, M.J., Padua, D., von Praun, C.: Programming for Parallelism and Locality with Hierarchically Tiled Arrays. In: PPoPP '06, 11th ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 48-57. ACM New York, NY, USA (2006).
10. Furtado, P., Baumann, P.: Storage of Multidimensional Arrays Based on Arbitrary Tiling. In: 15th International Conference on Data Engineering, pp. 480-489. IEEE (1999).
11. Begau, C., Sutmann, G.: Adaptive dynamic load-balancing with irregular domain decomposition for particle simulations. J. Computer Physics Communications, vol. 190, pp. 51-61. Elsevier B.V. (2015).
12. Fattebert, J.-L., Richards, D.F., Glosli, J.N.: Dynamic load balancing algorithm for molecular dynamics based on Voronoi cells domain decompositions. J. Computer Physics Communications, vol. 183, no. 12, pp. 2608-2615. Elsevier B.V. (2012).
13. Deng, Y., Peierls, R.F., Rivera, C.: An Adaptive Load Balancing Method for Parallel Molecular Dynamics Simulations. J. of Computational Physics, vol. 161, no. 1, pp. 250-263. Elsevier B.V. (2000).
14. Fleissner, F., Eberhard, P.: Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection. Int. J. for Numerical Methods in Engineering, vol. 74, no. 4, pp. 531-553. John Wiley & Sons, Ltd. (2008).
15. Hayashi, R., Horiguchi, S.: Efficiency of dynamic load balancing based on permanent cells for parallel molecular dynamics simulation. In: IPDPS 2000, 14th International Parallel and Distributed Processing Symposium, pp. 85-92. IEEE (2000).
16. Rope and Patch Demonstration Page, http://ssd.sscc.ru/en/algorithms.