Noname manuscript No.
(will be inserted by the editor)

Scalable Distributed Data Allocation in LuNA Fragmented Programming System

Victor Malyshkin · Vladislav Perepelkin · Georgy Schukin

Received: date / Accepted: date

Abstract The paper presents a scalable distributed algorithm for static and dynamic data allocation in LuNA fragmented programming system. LuNA is intended for automation of construction of parallel programs, which implement large-scale numerical models for multicomputers with large number of computing nodes. The proposed algorithm takes into account data structure of the numerical model implemented, provides static and dynamic load balancing and can be used with various network topologies.

Keywords scalable distributed system algorithm, dynamic data allocation, distributed algorithms with local interactions, fragmented programming technology, fragmented programming system LuNA

1 Introduction

Implementation of numerical models on multicomputers with large number of computing nodes is a challenging problem in the domain of high-performance parallel computing. To achieve good efficiency and scalability of parallel programs dynamic load balancing and/or effective resources allocation strategy is necessary. In the light of growing size of multicomputers (in terms of memory capacity, number of cores, etc.) new system algorithms for data processing

This work was supported by Russian Foundation for Basic Research (grants 14-07-00381 a and 14-01-31328 mol_a).

V. Malyshkin, V. Perepelkin, G. Schukin
Institute of Computational Mathematics and Mathematical geophysics SB RAS, Novosibirsk, Russia
E-mail: {malysh, perepelkin, schukin}@ssd.sscc.ru

V. Malyshkin, V. Perepelkin
Novosibirsk State University, Novosibirsk, Russia

V. Malyshkin, G. Schukin
Novosibirsk State Technical University, Novosibirsk, Russia

and computations organization are to be developed. To simplify construction of parallel programs, capable of achieving high performance, LuNA system [1–3] is being developed.

In LuNA an application algorithm is represented as two sets: a set of immutable data pieces (data fragments, DF) and a set of side-effect free computational processes (computational fragments, CF). Each DF is produced by one CF and may be used as an input by other CFs; each CF requires values of all its input DFs to be executed and is executed only once. In LuNA a program is executed by a run-time system, which performs DFs and CFs migration between processing elements (PEs, i.e. computing nodes of a multicomputer) in order to transmit input DFs to each CF and to equalize workload.

Efficiency of LuNA program execution (in terms of running time, memory consumption, etc.) depends on the quality of CFs and DFs distribution on PEs. In the paper authors propose a scalable distributed algorithm for dynamic DFs and CFs allocation, which is employed by LuNA system.

2 Related Works

The problem of efficient and scalable data allocation is actively researched. Worth mentioning are scalable diffusion-like algorithms ([6–8]), since they do not require global interactions, but they lack concerning data structures, balancing speed and tolerate global imbalance with low load gradient.

Data allocation problems are also faced in distributed databases ([12–15]) and cloud services ([16–18]). Due to relatively small number of objects to distribute and low migration rates, these systems take advantage of centralized algorithms, which are, although, not scalable to large number of allocation units (such as CFs or DFs), nor to multicomputers with large number of PEs.

Good efficiency can be achieved when allocating data of particular structures, such as meshes ([19, 20]). However, application domain of these algorithms is limited to these data structures.

Worth mentioning are static analysis algorithms, employed in compilers ([9–11]). Their limitation is static decision making (at compile time).

Such algorithms as [21, 22] do not take data structure into account and do not solve the problem of data search. In [15] a relatively scalable algorithm is presented, but it employs global communications, which should be avoided.

3 Requirements for Data Allocation Algorithm

In order to provide high efficiency and scalability of a parallel program, data allocation algorithm should:

- Provide approximately equal load of available PEs (static and dynamic load balancing) in terms of computational time, memory consumption, etc.
- Reduce amount of communications by taking data structure into account
- Tune to behavior of phenomena being modeled

- Be decentralized and mostly use short-distance communications

4 Distributed Algorithm of Data Allocation

4.1 Peculiarities of Data Allocation in LuNA System

In LuNA system at any time moment each DF and CF has a single PE assigned with it, called *residence*. DF residence is a PE, on which the DF value is stored. CF residence is a PE, on which the CF will be executed. Also, in LuNA each DF and CF are identified by unique (program-wide) identifiers.

One of the responsibilities of LuNA system is to deliver values of input DF of every CF to the residence of the CF. For each CF all identifiers of its inputs DFs are known. To enable DF lookup, LuNA run-time system transfers DF values to their residences, as well as provides an ability to search a residence by DF (or CF) id.

Dynamic load balancing in LuNA is a process of residence reassigning in order to equalize workload. Reassigning a residence causes DFs and CFs migration, which is also performed by LuNA run-time system.

4.2 Data Allocation Problem Formulation

Since the residence mechanism is the same for both CFs and DFs, only DFs are considered below. The problem of data allocation is formulated as follows. Let D be a set of all DFs in a program, and P be a set of all PEs. It is required to build a function $r : (D, T) \to P$, which dynamically assigns each $df \in D$ to its residence $p \in P$ for each time moment $t \in T$ (T denotes a set of all time moments of program execution and can be considered as $T = [0; 1]$, where 0 corresponds to execution begin, and 1 corresponds to execution end).

Consider an example. Let at time moment t=0 there is an initial DFs distribution $r_1 : D \to P$. At time moment $t' \in (0; 1)$ a workload imbalance was detected and balancing was performed, producing a new DFs distribution $r_2 : D \to P$, which was unchanged until the end of program execution. For this example function r would be defined as follows:

$$r(df, t) = \begin{cases} r_1(df), t < t' \\ r_2(df), t >= t' \end{cases}$$

4.3 The Proposed Algorithm of Data Allocation

It's hard to automatically construct a function r that will lead to a good data distribution, hence human assistance is beneficial. For that purpose definition of r is divided into two steps. **Firstly** a user manually defines an intermediate function $v : D \to V$, where V is a set of identical nodes of a virtual multicomputer. The virtual multicomputer is defined by its size (number of virtual

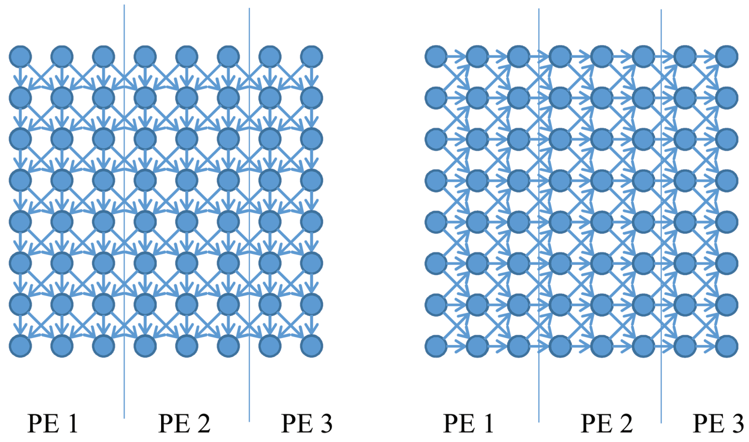

Fig. 1 Computation time (in seconds) dependency on number of PEs.

nodes) and has a 1D lattice network topology (mesh or torus). The function v assigns each DF a *virtual residence*, which is static, i.e. does not depend on time. **Secondly** a function $p : (V, T) \to P$ is defined, which dynamically assigns a physical PE to a virtual node. This function is dynamically defined by LuNA run-time system. In such a way, a residence of a DF df at a time moment t is defined as $r(df, t) = p(v(df), t)$.

To define function v a user has to manually solve a simplified problem of DFs distribution. The simplifications are the absence of dynamism and virtual multicomputer heterogeneity, and the linearity of the virtual network topology. Despite the fact that v definition may be not easy, it is still much easier than solving the original problem.

For function r to meet the requirements from section 3, function v should:

- Reduce neighborhood violation, i.e. total distance between virtual residences of a CF and its input DFs.
- Reduce load imbalance, i.e. total amount of computations each virtual node holds.

The criteria presented do not form the full system of requirements, yet they can be useful both to judge on quality of a given v and to get the idea of a good v. In particular, it is essential not only to reduce total load imbalance, but also to reduce load imbalance at each time moment. This point can be illustrated by a following simple example. Consider a 1D explicit finite-differential scheme. Due to immutability of DFs the set of DFs would be $D = \{df_{i,t}\}$, where i is a space coordinate, and t is a time coordinate. Both $v(df_{i,t}) = i$ and $v'(df_{i,t}) = t$ provide equal load for each virtual node, but in the first case all the nodes are able to operate in parallel, while in the second case only sequential execution is possible due to data dependencies (see Fig. 1).

Once v is defined, a virtual multicomputer is dynamically mapped to a physical multicomputer by LuNA run-time system, thus defining function p. Function p is constructed in accordance with physical network topology, i.e. two neighboring virtual nodes are assigned to the same PE or to PEs, neighboring in physical topology. This neighboring requirement is easy to satisfy for common network topologies, such as multi-dimensional lattice, fat tree or cluster. Also, p should be constructed in a way, when every PE receives approximately the same number of virtual nodes (and, thus, about the same amount of workload).

Definition of v as a separate intermediate step allows user to express his knowledge on how to efficiently distribute data. Based on this knowledge LuNA run-time system constructs actual distribution.

4.4 Dynamic Load Balancing

The proposed algorithm supports dynamic load balancing. The load balancing algorithm is based on a well-known diffusion-type dynamic load balancing. It is assumed that workload of PEs is periodically estimated or measured in some way, and sent to all neighboring PEs. Once difference of workloads of two neighboring PEs exceeds a threshold value, a balancing procedure, which transfers workload from the overloaded PE to the underloaded one, is started.

For the proposed data distribution algorithm transferring workload is performed as follows. Some of virtual nodes, currently mapped to the overloaded PE are reassigned to the underloaded PE. This causes reassigning of CF and DF residences and, as a consequence, CFs and DFs migration. Balancing must not violate the above-mentioned neighboring requirement (which is always possible).

4.5 Peculiarities of the Proposed Algorithm

Since load balancing is implemented as residences reassignment, the atomic unit of load balancing is a group of DFs and CFs, mapped to single virtual node. Therefore, number of virtual nodes has to be times greater than number of physical nodes in order to provide granularity for balancing.

5 Experiments

To investigate the efficiency of the proposed algorithm an explicit finite difference method for 3D Poisson equation solution [5] was chosen as a test application.

The experiments were conducted on MVS-10P cluster of Joint Supercomputer Center of RAS (each cluster node has two Xeon E5-2690 processors with 64 Gb RAM; nodes are connected by Infiniband FDR network). GCC 5.2.0 compiler and MPICH 3.1.4 communication library were used. Each test run

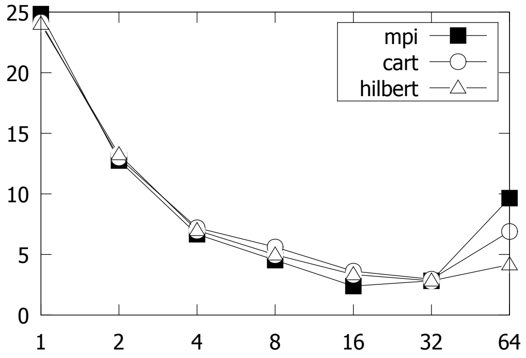

Fig. 2 Computation time (in seconds) dependency on number of PEs.

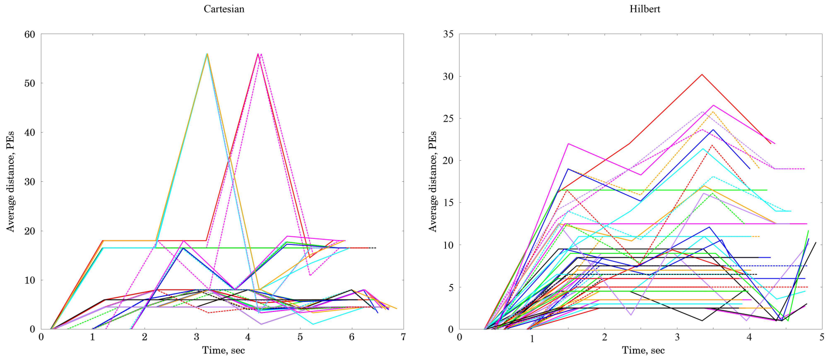

Fig. 3 Average DF communication distance at different time moments. Different colors denote different PEs.

consisted of 100 iterations of FDM on 400 × 400 × 400 mesh, decomposed into 8 × 8 fragments.

Two LuNA versions were tested: *hilbert* and *cartesian*. They differ in the way virtual nodes are mapped to physical PEs. The first version employs Hilber space-filling curve, while the second one uses naive row-by-row mapping. Also, for comparison a straightforward MPI implementation of the same method was developed.

In Fig. 2 total computation times are shown. All three implementations demonstrate approximately the same efficiency with a slight advantage of the hilbert version over the cartesian one. Fig. 3 demonstrates the average distance (in PEs hops) of DFs communication for given PE (different colors denote different PEs). The hilbert version is more efficient, providing approximately twice less distance then the cartesian one.

6 Conclusion

The problematics of data distribution automation for implementation of large-scale numerical models for supercomputers is considered. An algorithm for dynamic data allocation for LuNA fragmented programming system is proposed. Performance tests of the algorithm are presented. The proposed algorithm is designed for large multicomputers (thousands of PEs and more). It is scalable, allows dynamic workload balancing and considers supplementary information on application data structure. Further research is to be focused on overcoming limitations of the algorithm: static nature of the supplementary information and single dimension of neighborhood description.

References

1. 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)
2. 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)
3. 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)
4. 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)
5. Kireev, S.E., Malyshkin V.E.: Fragmentation of Numerical Algorithms for Parallel Subroutines Library. J. Supercomputing, vol. 57, no. 2, pp. 161–171 (2011)
6. Kraeva, M.A., Malyshkin, V.E.: Dynamic Load Balancing Algorithms for Implementation of PIC Method on MIMD Multicomputers. J. Programmirovanie, no. 1, pp. 47–53 (in Russian) (1999)
7. Hu, Y.F., Blake, R.J.: An Improved Diffusion Algorithm for Dynamic Load Balancing. J. Parallel Computing, vol. 25, no. 4, pp. 417–444 (1999)
8. Corradi, A., Leonardi, L., Zambonelli F.: Performance Comparison of Load Balancing Policies Based on a Diffusion Scheme. In: Euro-Par'97 Parallel Processing, pp. 882–886. Springer, Heidelberg (1997)
9. Anderson, J.M., Lam, M.S.: Global Optimizations for Parallelism and Locality on Scalable Parallel Machines. In: ACM-SIGPLAN PLDI'93, pp. 112–125. ACM New York, USA (1993)
10. Li, J., Chen, M.: The Data Alignment Phase in Compiling Programs for Distributed-Memory Machines. J. Parallel and Distributed Computing, vol. 13, no. 2, pp. 213–221 (1991)
11. Lee P.: Efficient Algorithms for Data Distribution on Distributed Memory Parallel Computers. J. IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 8, pp. 825–839 (1997)
12. Yu-Kwong Kwok, Ahmad, I.: Design and Evaluation of Data Allocation Algorithms for Distributed Multimedia Database Systems. IEEE Journal on Selected Areas in Communications, vol. 14, no. 7, pp. 1332–1348. IEEE (1997)
13. Iacob, N.M.: Fragmentation and Data Allocation in the Distributed Environments. Annals of the University of Craiova - Mathematics and Computer Science Series, vol. 38, no. 3, pp. 76–83 (2011)
14. Jagannatha, S., Geetha, D.E., Suresh Kumar, T.V., Rajani Kanth, K.: Load Balancing in Distributed Database System using Resource Allocation Approach. J. Advanced Research in Computer and Communication Engineering, vol. 2, no. 7, pp. 2529–2535 (2013)

15. Honicky, R.J., Miller E.L.: Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution. In: 18th International Parallel and Distributed Processing Symposium (2004)
16. Alicherry, M., Lakshman, T.V.: Network Aware Resource Allocation in Distributed Clouds. In: INFOCOM 2012, pp. 963–971. IEEE (2012)
17. AuYoung, A., Chun, B.N., Snoeren, A.C., Vahdat, A.: Resource Allocation in Federated Distributed Computing Infrastructures. In: First Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure (2004)
18. Raman, R., Livny M., Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing. J. Cluster Computing, vol. 2, no. 1, pp. 129–138 (1999)
19. Reddy, C., Bondfhugula, U.: Effective Automatic Computation Placement and Data Allocation for Parallelization of Regular Programs. In: 28th ACM International Conference on Supercomputing, pp. 13–22. ACM New York, USA (2014)
20. Baden, S.B., Shalit, D.: Performance Tradeoffs in Multi-tier Formulation of a Finite Difference Method. In: ICCS 2001, LNCS, vol. 2073, pp. 785–794. Springer, Heidelberg (2001)
21. Ken-ichiro Ishikawa. ASURA: Scalable and Uniform Data Distribution Algorithm for Storage Clusters. Computing Research Repository, abs/1309.7720 (2013)
22. Chawla, A., Reed B., Juhnke, K., Syed, G.: Semantics of Caching with SPOCA: A Stateless, Proportional, Optimally-Consistent Addressing Algorithm. In: USENIX Annual Technical Conference 2011, pp. 33–33. USENIX Association (2011)
23. Lowder, J.K., King, P.J.H.: Using Space-Filling Curves for Multi-dimensional Indexing. In: Advances in Databases, LNCS, vol. 1832, pp. 20–35. Springer, Heidelberg (2000)
24. Moon, B., Jagadish, H.V., Faloutsos, C., Saltz, J.H.: Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. J. IEEE Transactions on Knowledge and Data Engineering, vol. 13, no. 1, pp. 124–141 (2001)