Control Flow Usage to Improve Performance of Fragmented Programs Execution

V.E. Malyshkin[1,2,3], V.A. Perepelkin[1,2], A.A. Tkacheva[1,2]

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

Abstract. Dataflow-based systems of parallel programming, such as LuNA fragmented programming system, often lack efficiency in high performance computations due to a high degree of non-determinism of a parallel program execution and execution overhead it causes. The authors concern defining control flow in LuNA programs in order to optimize their execution performance. The basic idea is to aggregate several fragments of a program and to execute them under control flow, thus reducing both surplus parallelism and system overhead. Tests presented show effectiveness of the proposed approach.

Keywords: High performance computing, fragmented programming technology, LuNA fragmented programming system, control flow

1 Introduction

Implementation of large-scale numerical models on supercomputers is often challenging, because a programmer has to develop a program, possessing a number of dynamic properties, such as dynamic load balancing and tuning to available resources in order to make the program efficient and scalable. To increase the accessibility of supercomputers, a variety of parallel programming systems and tools were developed. To hide low-level programming details they automate the provision of the dynamic properties of programs. Examples of such systems are: LuNA [1], Charm++ [2], SMP Superscalar [3], DPLASMA [4].

High level of a program brings forth problems of its efficient parallel execution on a supercomputer. Since a programming system is able to execute the program in many ways (non-determinism of a program), appropriate to different hardware configurations or input data, the system faces the problem of dynamic choice of an efficient way of the program's execution. Often being NP-complete, this problem has to be overcome with heuristics or particular solutions.

One of the common ways to improve parallel program execution performance for such dataflow-based systems as LuNA is control flow usage. Control flow can presume for a number of subtasks of a program the order of their execution statically,

eliminating the need to track data dependencies in run-time. This reduces the run-time system overhead, thus improving the efficiency of the program execution. Although the parallelism degree of the program also decreases, this is often an advantageous trade-off.

The aim of the current work is the development of control flow support for LuNA programming system.

2 Related Works

Control flow means are widely used to improve performance of declarative parallel programming languages and systems.

In the systems SMP Superscalar [3], Cell Superscalar [5], ProActive Parallel Suite [6], one can define control flow with priorities. This mechanism, while being flexible, is not capable of defining complex program behavior. It also lacks readability for large programs, which results in control errors probability increase. In the functional languages of parallel programming Haskell [7], SISAL [8] in order to improve performance of algorithm's execution there are dedicated language means to mark a loop as parallel or sequential.

The library DPLASMA for linear algebra subroutines for dense matrices is built over PaRSEC [9] system. This system analyses informational dependencies at compile time, and the algorithm is represented in the Directed Acyclic Graph (DAG) form.

In such a way, while the idea is widely exploited, no general to construct control flow has been developed, thus further research of the problem is necessary.

3 LuNA Fragmented Programming System

LuNA (Language for Numerical Algorithms) is a language and a parallel programming system [1] intended for implementation of large-scale numerical models on supercomputers. It is being developed in the Institute of Computational Mathematics and Mathematical Geophysics of the Siberian Branch of Russian Academy of Sciences.

In LuNA an application algorithm is represented in a single-assignment coarse-grained explicitly parallel language LuNA as a bipartite graph of *data fragments* (DF) and *computational fragments* (CF). DFs are basically blocks of data (submatrixes, array slices, etc.). CFs are applications of pure functions on DFs. A CF has a number of input DFs and a number of output DFs. Values of output DFs are computed by the CF from the values of input DFs. Such representation is called *fragmented algorithm* (FA).

LuNA program consists of the FA description in LuNA language and a dynamic load library with a set of conventional sequential procedures. CFs are implemented as calls to these procedures with input and output DFs. Execution of all the CFs is done in accordance with partial order, that is imposed on the set of CFs by the information dependencies, forms the FA execution.

A FA is executed by the LuNA run-time system. Fragmented structure of the FA is kept in run-time, allowing the run-time system to dynamically assign CFs and DFs to different computing nodes, execute CFs in parallel (if possible), balance computational workload by redistributing CFs and DFs and so on.

Since the run-time system makes most decisions on FA execution dynamically, many checks for CFs being ready take place. This leads to significant overhead.

4 Suggested Approach

The basic idea of the suggested approach is to combine a subset of CFs into a single CF. For example, a loop or a subroutine may be collapsed into a single CF. All DFs, necessary to compute the CF, become input DFs, and all DFs produced become its outputs. All the CFs combined are executed sequentially in a fixed order under the control flow. Also, the number of CFs reduces, causing reduced system overhead.

Such FA transformation is possible for a set of CFs if the following condition is true: once all input DFs are provided, the whole chain of CFs may be sequentially executed without having to wait for other CFs to execute.

The transformation may be applied to both dependent and independent CFs. If the latter is the case, then the overall degree of parallelism of the FA is reduced, therefore control flow usage should not be overused.

To determine permissible order of CFs execution information dependencies analysis must be made. Creation of new aggregated CF means generation of new procedure, which invokes other procedures in established order. Both dependencies analysis and procedure generation were implemented as a part of LuNA project for a particular case, i.e. when the part of FA being transformed contains statically defined set of CFs.

It is worth mentioning, that it is algorithmically hard to determine effectiveness of control flow optimization in a given case. This problem is out of the scope of this work. Instead, the LuNA language was extended with annotations, which explicitly define, whether control flow transformation is required or not for given part of FA. Currently the annotations are provided by a user, but annotations generation may be automated.

5 Performance Tests

To investigate the effectiveness of the suggested approach a number of performance tests were conducted on a shared memory multiprocessor with 6 cores (Intel Xeon CPU X5600 2.8 GHz) and on the distributed memory cluster of National Research University of Novosibirsk [10].

An application tested is a typical reduction problem on an array of numbers. The array to reduce is split into a number of slices. Each slice is reduced sequentially, while different slices may be processed independently. Two tests were performed: in the first one the slices were reduced sequentially, in the second one the slices were reduced in parallel. Both sequential and parallel tests were performed with and with-

out control flow optimization. Control flow was applied to all the slices. Slice size was a parameter of the tests.

In all the tests “LuNA” stands for non-optimized execution and “static” stands for execution with control-flow optimization.

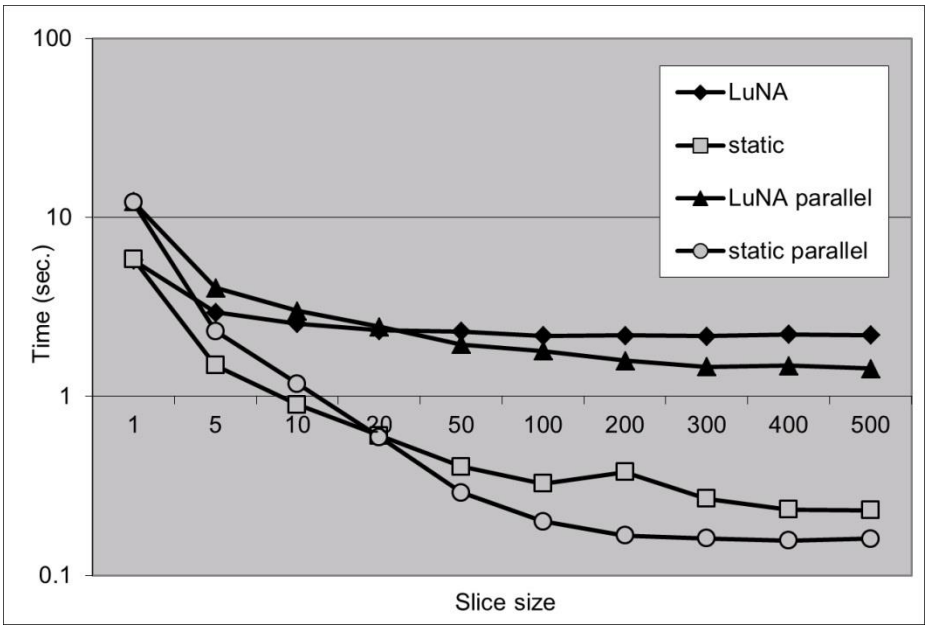

Fig. 1. Multiprocessor test. Dependency of the FA execution time on the slice size is shown. Array size is 3×10^4.

From fig. 1 it is seen that control flow significantly reduces the FA execution time on the multiprocessor. The greater the slice size is, the more the benefit is. One can also notice, that for small sizes parallel version executes slower. This is due to extra overhead, which arises from extra CFs in parallel version of the test, and is not relevant to the subject of the work.

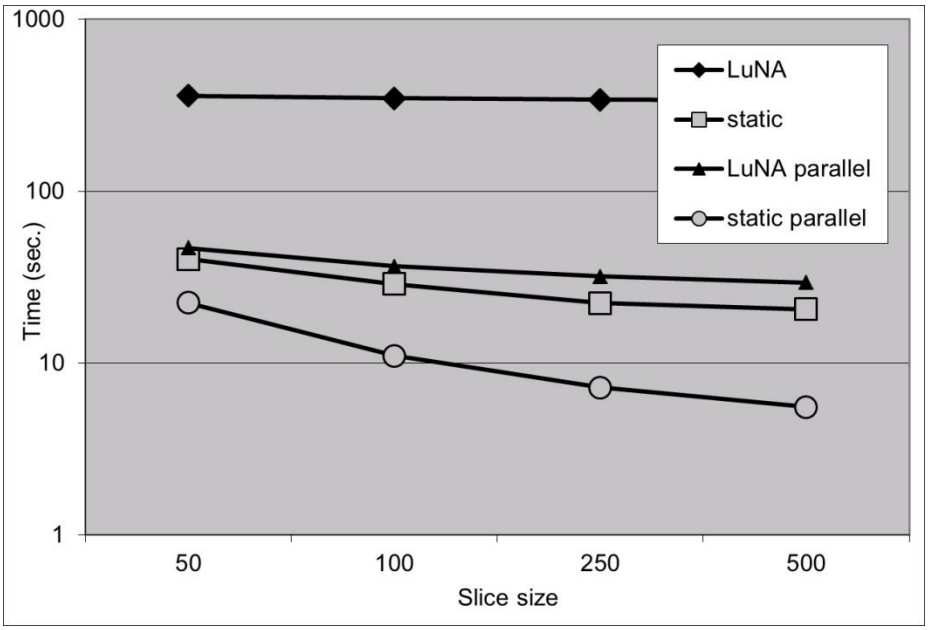

Fig. 2. Multicomputer test. Dependency of the FA execution time on the slice size is shown. Array size is 5×10^5. Number of computer nodes is 8.

Figure 2 shows the result of the similar testing on the distributed memory cluster. It is seen, that usage of direct control benefits in performance by reducing system overhead.

6 Conclusion

Profitability of direct control usage in LuNA fragmented programming system was concerned. An approach to employ direct control in LuNA system was suggested and implemented as a part of LuNA software. Performance tests demonstrate that control flow usage significantly increases performance of program execution on the example of the reduction problem.

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

References

1. Malyshkin, V.E., Perepelkin, V.A.: LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. Proc. of the 11-th Conference on Parallel Computing Technologies, LNCS 6873, Springer pp. 53–61. (2011)

2. Acun, B. et al. Parallel Programming with Migratable Objects: Charm++ in Practice. ser. SC'14, New.York, NY, USA: ACM (2014)
3. Perez J.M., Badia R.M., and Labarta J. A flexible and portable programming model for SMP and multi-cores. Technical Report 03/2007, Barcelona Supercomputing Center, Barcelona, Spain, (2007)
4. Bosilca, G. et al. Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA. Proc. of the Workshops of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2011 Workshops), IEEE, Anchorage, Alaska, USA, 1432-1441 (2011)
5. Bellens, P., Perez, J.M., Badia, R.M., Labarta, J. CellSs: a programming model for the Cell BE architecture. In SC '06: Proc. of the 2006 ACM/IEEE conference on Supercomputing, p 86, New York, NY, USA, ACM Press (2006)
6. Caromei, D., Leyton, M., ProActive Parallel Suite: From Active Objects-Skeletons-Components to Environment and Deployment. Euro-Par 2008 Workshops – Parallel Processing, pp. 423-437 (2008)
7. Coutts, D., Loeh, A. Deterministic Parallel Programming with Haskel. Computing in Science and Engineering, vol. 14, no. 6, pp. 36-43,(2012)
8. Gaudiot, J., DeBoni, T., Feo, J., Boehm, W., Najjar, W., Miller, P. Sisal Project: Real World Functional Programming. LNCS 1808, Springer-Verlag, pp. 45-72 (2001)
9. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., Dongarra, J. PaRSEC: Exploiting Heterogeneity to Enhance Scalability. IEEE Computing in Science and Engineering, Vol. 15, No. 6, 36-45. (2013)
10. National Research University of Novosibirsk cluster. http://www.nusc.ru/