An Effective Implementation Approach for Adapting of HPC Applications

Pavel Drobintsev, Vsevolod Kotlyarov, Aleksei Levchenko, and Alexey Maslakov

Peter the Great St. Petersburg Polytechnic University, Saint Petersburg, Russia
vpk@spbstu.ru

Abstract. The work considers an approach to the adaptation of applications that use algorithms proven to be successful and developed with the help of computers that do not have a high degree of parallelism, though in a number of implementations they require a sharp reduction in the computing time. A natural way-out is to transfer the algorithm solution to a highly parallel heterogeneous processing environment, i.e. hybrid supercomputer. Unfortunately, the result does not always meet expectations. The challenge is the need to consider architectural features of the supercomputer and the corresponding translation of the generic algorithm, while maintaining its semantic features, i.e. the development of parallel software of the generic algorithm scalable to allocated supercomputer resources. Available approaches to the software parallelization deliver superb results when algorithms demonstrate obvious parallelism. Otherwise, their transformation to the parallel representation requires an analysis of dependencies in parallel threads on data and costs of the parallel supercomputer execution. In this paper, we present an algorithm analysis technique that allows to determine fragments for a significant reduction in the computing time during the parallel execution. The result is a algorithm specification work schedule that ensures the effective solution, using the supercomputer. The schedule is used to create the dedicated control over the execution of an parallelized algorithm for its effective solution with the help of hybrid supercomputer resources. The work shows results of the implementation of the developed technique in terms of the genetic research.

Keywords: computer architecture · high performance computing · Java · large-scale systems · NUMA · performance

1 Introduction

There are currently a variety of areas of concern, where processing and analysis of large data arrays are actively used. For example, these include bioinformatics, photograph-based modeling of the surface relief, analysis of data on industrial plots for the optimization of manufacturing processes, etc. These areas have applications with algorithms that have proven to be successful in data processing of significantly smaller arrays, developed with the help of computers that do not have a high degree of parallelism. As a rule, capacities of such algorithms do not meet modern requirements for processing of huge data arrays and a sharp reduction in the time of the computation result output.

This work considers an adaption approach to applications that use successfully tested algorithms for the highly-parallel processing environment, namely the supercomputer. Unfortunately, the direct transfer result does not always meet expectations. The challenge is the need to consider architectural features of the supercomputer and the corresponding translation of the generic algorithm, while maintaining its semantic features, i.e. the development of parallel software of the source algorithm scalable to allocated supercomputer resources.

2 Related Work

When analyzing available papers devoted to the software adaptation to hybrid supercomputer clusters, certain problems specific to the hybrid cluster application programming model shall be considered:

- a problem of the resource allocation dynamic control,
- a problem of the task execution control especially in heterogeneous clusters,
- a problem of the software adaptation for the effective execution in several clusters based on the required load.

There are well-known publications devoted to the solution of these tasks. Work [11] is one of the earliest to offer a guided self-scheduling that allows to ensure the workload balance. Work [13] offers a export strategy of information about used supercomputer resources for task-planning at workstations and simultaneously in available cluster nodes. Scogland et. al [12] proposes methods and systems that allow the software to get adapted to heterogeneous computing systems directly at runtime. First and foremost, the adaptation includes the load balancing and ensures the cache coherence. Recent work [10] describes an approach to use a parallel scripting language to run scientific supercomputer applications across multiple computing resources, ranging from the academic university cluster to TOP500 supercomputers. Another work [17] describes an efficient resource manager on the basis of the application execution schedule in the supercomputer, which has allowed a 40% reduction in the task execution wait time if compared to classic solutions like Slurm [15]. These publications and some other papers [7, 14, 8] consider the management of applications as that of cluster resources and the distribution of these resources across computing systems of different types.

This work presents an approach to the application adaptation based on its preliminary static analysis and the generation of an effective execution schedule in view of the architecture of the hybrid supercomputer and application requirements for cluster resources.

3 Adaptation Approach

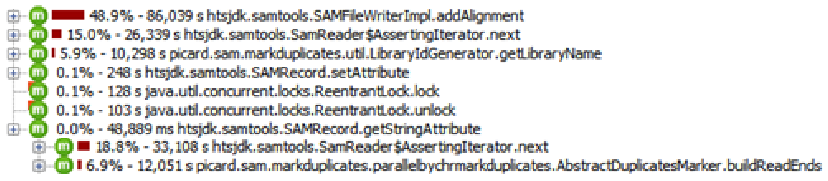
The specific contribution of this paper is an approach that allows to run Java parallel applications across completely different HPC systems (hybrid supercomputer). As such, we developed a parallel version of MarkDuplicates, a bioinformatics tool from Picard toolkit [1] used in the genome analysis to mark PCR duplication artifacts in RNA-Seq data that is stored in giant (multi terabyte) BAM or SAM files. To ensure fast input/output, these files should be stored and processed directly in-memory of one node. The developed parallel MarkDuplicates is an example of irregular and unbalanced consumption of HPC resources by the nature of its task: some modules perform computational tasks, while other modules periodically require connection to the input files.

With regard to the principal differences between our HPC systems, that caused the development of our approach, the computational part is performed on a traditional Linux cluster, while big data processing occurs directly in the global shared memory of the ccNUMA system. The division of tasks between systems is caused by complexity in providing Java support for huge pages and logically indivisible piece of RAM more than 2Tb. It is important to note that despite the variety of parallel computing models used in HPC, such as the traditional MPI+X paradigm or emerging parallel programming models like UPC/Charm++/X10, etc., we are forced to use Java for big data processing in this case because of existing Java codebase in bioinformatics. At the same time, we were not able to store terabyte files in the memory of the standard cluster node, which is equipped with only 60-256Gb RAM. We could use traditional not-in-memory network storage, but the interconnect latency ruins the performance gain, because the communication operations far outweigh the computational ones. The results of applying the developed approach to our systems will also be mentioned in Section 6. Thus we perform computational operations and processing of large data in hybrid environment of different systems, depending on their suitability. This approach has significant advantages since in-memory data processing is still the fastest pathway.

The above circumstances prompted us to develop the following procedure to analyze the algorithm in order to search for program modules that will give a significant reduction in the computing time for the parallel execution, corresponding to the program correction that increases the degree of parallelism and creates the execution schedule to ensure the effective solution with the help of the supercomputer. The procedure stipulates the following stages of analysis:

1. *Call tree analysis of algorithm methods* obtained as a result of profiling of an unadapted MarkDuplicates version to determine approaches, whose call percentage is significant in the application. Figure 1 illustrates the result of the tree analysis, which is the identification of a list of methods with the highest frequency of calls. There are 4 methods in the application under consideration: SAMFileWriterImpl.addAlignment (48.9%), SamReader$AssertingIterator.next (33.8%), buildReadEnds (6.9%) and getLi-

braryName (5.9%). The contribution of remaining used methods is insignificant in terms of parallelization.



```
48.9% - 86,039 s htsjdk.samtools.SAMFileWriterImpl.addAlignment
15.0% - 26,339 s htsjdk.samtools.SamReader$AssertingIterator.next
5.9% - 10,298 s picard.sam.markduplicates.util.LibraryIdGenerator.getLibraryName
0.1% - 248 s htsjdk.samtools.SAMRecord.setAttribute
0.1% - 128 s java.util.concurrent.locks.ReentrantLock.lock
0.1% - 103 s java.util.concurrent.locks.ReentrantLock.unlock
0.0% - 48,889 ms htsjdk.samtools.SAMRecord.getStringAttribute
  18.8% - 33,108 s htsjdk.samtools.SamReader$AssertingIterator.next
  6.9% - 12,051 s picard.sam.markduplicates.parallelbychrmarkduplicates.AbstractDuplicatesMarker.buildReadEnds
```

Fig. 1. A segment of the call tree of MarkDuplicates metric methods

2. *The algorithm correction through (initial or additional) parallelization of commonly used methods.* During parallelization, dependencies in terms of data and control in parallel threads shall be considered with the provision of the proper synchronization. There are well-known computerized approaches to the search and analysis of dependencies [6, 5].
3. *Analysis of parallel threads that implement the application.* Indices of CPU total usage and the number of active threads shown in the diagram in figure 2 are divided into five areas, whose needs vary from 28 threads for area 1 to 7 threads for area 5, in view of 5 system service threads. Since a core is allocated to each thread, unused cores remain idle in areas 1–5.

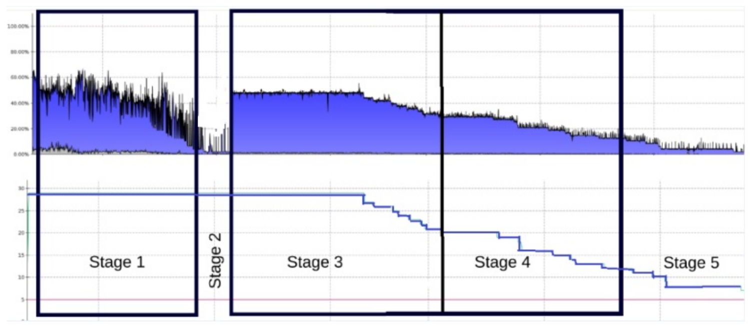


Fig. 2. Stages of parallel threads activity during MarkDuplicates execution

4. *Evaluation of the thread-required memory space.* An example of the assessment of the thread-required memory space for the solution of parallel algorithm fragments is shown in figure 3. The evaluation is obtained as a result of profiling of the application execution in the supercomputer. Finally, if during the execution the thread-required memory space exceeds the allocated resource, for heap or cache are generated requirements for the additional

algorithm fragmentation for the solution of each thread to fit into allocated resources.

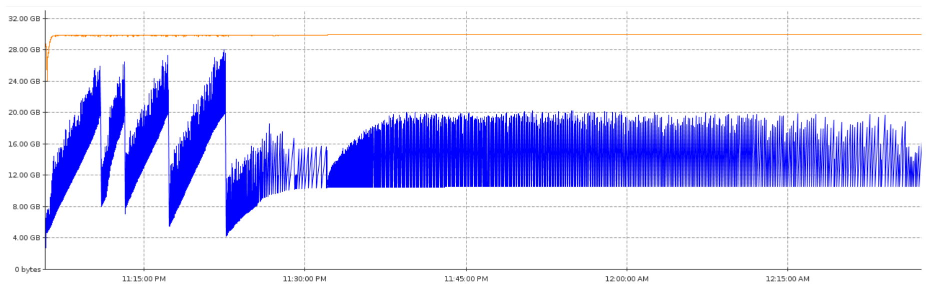


Fig. 3. Diagram of requirements for the thread memory when executing the application

5. *Creation of an implementation application model for the supercomputer.* Figure 4 shows the model of application requirements for resources. Since the model takes into account requirements for the effective implementation of a particular application, it shall be transformed into a schedule for its implementation in the supercomputer. In this event, the application is transformed into a task package, each of which carries information about required resources.

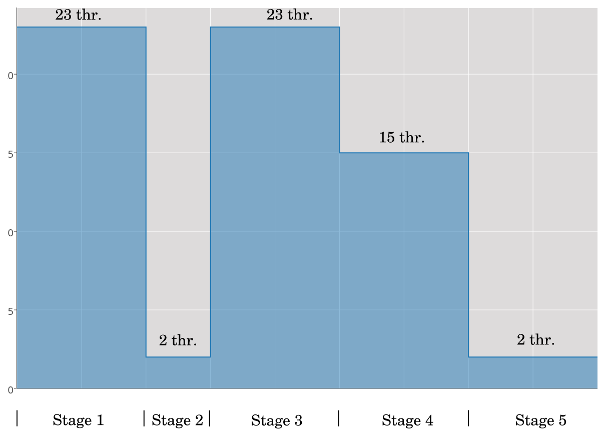


Fig. 4. Model of application requirements for resources

4 Running Application on Hybrid Supercomputer

There are a lot of well-known papers devoted to the task thread management in high performance computing (HPC) [8, 14, 7]. What is peculiar about them is that they offer universal approaches for the successful solution of described problem. The specifics of the solution proposed in the work is to implement special control adapted for a certain application. The similar approach is particularly relevant for the hybrid supercomputer, whose clusters are ready to provide resources both to applications with high demands for the speed-of-response and to applications that require huge memory space, or a combination of such requirements. Though available applications integrally possess similar features, their subtasks are characterized by more detailed requirements for the runtime environment, and they can combine the effective execution in different clusters of the hybrid supercomputer. An execution schedule is a must-have for such patterns.

The schedule formalization can be carried out in a number of ways. The work gives preference to the use of standardized control language UCM [9]. The language provides control scripts in the form of flow graphs loaded with information about requirements for resources. Each node of the control graph is associated with a specific task determined through the analysis of the application solution algorithm. Though UCM language is well-adapted for description successive control scenarios, scenarios with alternatives and scenarios with iterations are used so far for the control description. An example of the control scenario for the usage example study is shown in figure 5 (only requirements for threads and the memory are taken into account).

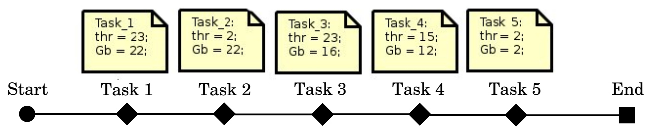


Fig. 5. Example of a schedule with a successive task control scenario

The schedule allows the task scheduler to distribute tasks both across resources of one cluster, and across resources of different clusters of the hybrid supercomputer.

5 Restrictions to be Considered When Planning Resources

An important feature of the efficient solution of parallel applications under study is the opportunity to use different node types of the hybrid supercomputer, depending on the stage of data processing and varying costs associated with the processing. Different operation stages of the same scientific application include

(1) compute-bound tasks, whose arithmetic operations greatly exceed the volume of communication operations, (2) bandwidth-bound tasks with predictable and successive access to the memory, (3) latency-bound tasks. The last type of tasks is characterized by high requirements for the size of the logically monolithic space of the globally addressable memory, intensive irregular addresses to the memory and poor spatial and temporal localization. The cost of the last task type dictates the need for a substantial increase in the memory of a single computing node if compared with pure computing tasks. This circumstance is stipulated by the processing task of large indivisible data because the computing system shall simultaneously meet requirements of parallel applications with high-computing intensity and requirements of applications from the big-data domain with high overhead costs of communications. To overcome restrictions of the memory space of one node in the hybrid supercomputer, it is possible to use multicomputer nodes with NUMA architecture (Non-Uniform Memory Access) equipped with RAM up to logically indivisible 12Tb for storage and processing of data used by MarkDuplicates application.

The memory non-uniformity inevitable in this case creates complex problems that shall be considered when adapting a parallel application to avoid sharp performance degradation. The main problems include (1) a computing node cache coherence problem, (2) a memory hot-spotting problem, (3) a false sharing problem. Furthermore, in case of Java-application for large data processing, difficulties may arise in handling memory space exceeding 12Tb, as a result of non-uniform remote access when working with large memory pages.

The cache coherence problem [2] is aggravated when substantial enlargement of hybrid supercomputer nodes is applied. The nodes are in charge of processing and storage of large data arrays directly in RAM. The computing process localization is determined by the algorithm feature to be divided into such modules, when the computing process of each of them is solved within the limits of its allocated resources: the number of cores, cache and memory module space. The contention for information from another cores cache deteriorates the localization, and the associated increase in the cache access time may disrupt the coherence [2]. This problem can be caused by (1) collective use of recorded data, (2) thread migration, (3) input-output tasks [4], i. e. all tasks actively used in applications. It is worth mentioning that non-localized false sharing of the same cache blocks by processors results in a runtime increase up to 8–10 times. Memory levels (DRAM, cache) and processing units form a hierarchy of locality groups (lgroups). Each group includes a set of processors and memory modules that are close to each other. In the case of hybrid supercomputer, such groups are multimachine nodes. The algorithm shall be designed so that on the appropriate phase interaction of data was localized in the appropriate thread, and exchange between threads happened at the time of their synchronization. In addition, garbage collection algorithm must be designed with these features for copying and compacting of active data objects when scaling across the hybrid cluster. In addition, the article [5] presented an approach to reduce the gap between memory latency and processor speed by improving data locality by resorting data for processing threads.

Hot-spot memory contention [3] is another complex problem. Large-scale multicomputer nodes of the hybrid supercomputer tend to get united and enlarged due to the generalization of RAM space, which provokes massive competitive access of thousands of competing processing units to critical data structures to the memory module, where the data are located. When developing a parallel version of the application, NUMA effects shall be considered, as well as the competition effect in access to the memory through a number of program mitigation methods, namely through (1) data redistribution and (2) software combining [3, 16]. Mitigation methods offer the decomposition of the algorithm and data into modules localized within memory borders attached to the supercomputer node. At the same time, algorithm modules are evenly distributed across different memory modules, which, in some cases, helps reduce the number of delayed memory addresses from 0.2 to 1.5, and the remote access delay to lowest possible values.

False sharing in a multi-threaded Java-application means that there is addressing to different objects that share a common cache memory block. False sharing effects can be mitigated with the help of the distribution based on the object creation time. However, when it comes to global shared memory nodes whose number of cores exceeds 3000, it can not be guaranteed that this effect will be completely avoided.

6 Results of Applying the Developed Approach

Experimental results of the proposed technique are shown using MarkDuplicates metric computation are given in the table 1 for the solution of serial (not parallelized) and scalable algorithm versions in Java. JVM Java HotSpot 64-bit Server VM in 25.102-b14 build for linux-amd64 have been used in the computation. We ran our experiments across multicluster hybrid supercomputer, which includes (1) global shared memory system (cc-NUMA architecture) and (2) RSC Tornado cluster, installed at Supercomputer Center, SPbPU. We used multimachine cc-NUMA macronodes with up to $\approx$12Tb of RAM for storing input files directly to memory. The second system, Tornado cluster, is equipped with a 1336 CPUs (Intel Xeon E5-2697 v3) and is used primarily for high-performance data processing. Computation time spans for the scalable version managed by the schedule at the solution in 1 and 2 clusters, and the serial (not parallelized) version were compared. Time spans were obtained during tests in the processing of data, whose array exceeds 200Gb. The resulting one-order difference demonstrates advantages of the proposed approach.

Similar results were obtained for a wide range of genetic algorithms. It should be noted that the C++ implementation of genetic algorithms and the use of OpenMP/MPI would lead to much better results, but biologists prefer Java, and formal code transformation without regard to problem semantics is unpromis-

Table 1. Experimental results

MarkDuplicates versions	**Time**$_{average}$	**Time**$_{max}$
Scalable version, data 234Gb (cluster No. 1)	298 min	302 min
Scalable version, data 234Gb (cluster No. 2)	110 min	114 min
Serial version, data 234Gb (cluster No. 1)	1123 min	1134 min

ing. Finally, it should be noted that the acceleration of genetic research is an important factor for the results application in the clinical practice.

7 Conclusion

The scope of the proposed technique application is, in the first place, processing algorithms for large data because the analysis of algorithms is usually costly and cannot be fully computerized. Therefore, the application of the technique is justified for repeatedly-used algorithms. For example, algorithms of genetic studies that should be quickly carried out in the medical practice. The technique is still studied, and its real application is so far limited to search and n-wise algorithms in C++ and Java,for which there are sufficiently powerful profilers. Nevertheless, it shows good results in the adaptation of algorithms created regardless of solution specifics in a highly parallel supercomputer environment.

The main drawback of the technique is its labor-intensive implementation. Therefore, it is planned in the future to computerize analysis tools for application algorithms to facilitate their better parallelization, to computerize the generation algorithm of schedules based on architectural features and resources of the execution environment. Create a single in-process chain and appropriate tools to computerize the adaptation of algorithms for the efficient execution in the hybrid supercomputer [5].

Acknowledgments. This work was financially supported by the Ministry of Education and Science of the Russian Federation within the framework of the state assignment № 29517.2017/64 (the project theme "Methods and technologies for verification and development of software for modeling and calculations using HPC platform with extramassive parallelism"). The results were obtained using ccNUMA system and RSC Tornado cluster in Supercomputer Center of Peter the Great St.Petersburg Polytechnic University.

References

1. Broad institute. Picard Toolkit (2017), https://github.com/broadinstitute/picard
2. Censier, L.M., Feautrier, P.: A new solution to coherence problems in multicache systems. IEEE Transactions on Computers C-27(12), 1112–1118 (Dec 1978)
3. Dandamudi, S.P.: Reducing hot-spot contention in shared-memory multiprocessor systems. IEEE Concurrency 7(1), 48–59 (Jan 1999)

4. Dubois, M., Scheurich, C., Briggs, F.A.: Synchronization, coherence, and event ordering in multiprocessors. Computer 21(2), 9–21 (Feb 1988)
5. Eimouri, T., Kent, K.B., Micic, A.: Effects of false sharing and locality on object layout optimization for multi-threaded applications. In: 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). pp. 1–5 (May 2016)
6. Ganesan, K., Chen, Y.M., Pan, X.: Scaling Java virtual machine on a many-core system. In: 2014 International Symposium on Integrated Circuits (ISIC). pp. 336–339 (Dec 2014)
7. Iserte, S., Prades, J., Reano, C., Silla, F.: Increasing the performance of data centers by combining remote GPU virtualization with Slurm. In: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). pp. 98–101 (May 2016)
8. McLay, R., Schulz, K.W., Barth, W.L., Minyard, T.: Best practices for the deployment and management of production HPC clusters. In: 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC). pp. 1–11 (Nov 2011)
9. Mussbacher, G., Amyot, D.: Assessing the applicability of use case maps for business process and workflow description. In: 2008 International MCETECH Conference on e-Technologies (mcetech 2008). pp. 219–222 (Jan 2008)
10. Ozik, J., Collier, N.T., Wozniak, J.M., Spagnuolo, C.: From desktop to large-scale model exploration with Swift/T. In: 2016 Winter Simulation Conference (WSC). pp. 206–220 (Dec 2016)
11. Polychronopoulos, C.D., Kuck, D.J.: Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. IEEE Transactions on Computers C-36(12), 1425–1439 (Dec 1987)
12. Scogland, T.R.W., Feng, W.C.: Runtime adaptation for autonomic heterogeneous computing. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 562–565 (May 2014)
13. Smallen, S., Crine, W., Frey, J., Berman, F., Wolski, R., Su, M.H., Kesselman, C., Young, S., Ellisman, M.: Combining workstations and supercomputers to support grid applications: the parallel tomography experience. In: Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat. No.PR00556). pp. 241–252 (2000)
14. Varrette, S., Bouvry, P., Cartiaux, H., Georgatos, F.: Management of an academic HPC cluster: The UL experience. In: 2014 International Conference on High Performance Computing Simulation (HPCS). pp. 959–967 (July 2014)
15. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: Simple Linux Utility for Resource Management, pp. 44–60. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
16. Zhang, M., Gu, N., Ren, K.: Optimization of computation-intensive applications in cc-NUMA architecture. In: 2016 International Conference on Networking and Network Applications (NaNA). pp. 244–249 (July 2016)
17. Zheng, X., Zhou, Z., Yang, X., Lan, Z., Wang, J.: Exploring plan-based scheduling for large-scale computing systems. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER). pp. 259–268 (Sept 2016)