PARUS: a parallel programming framework for heterogeneous multiprocessor systems

Alexey N. Salnikov (salnikov@cs.msu.su) and
Alexander S. Ivanov (sasha@cs.msu.su)

Moscow State University Faculty of Computational Mathematics and Cybernetics,
119992 Moscow, Leninskie Gory, MSU, 2-nd educational building, VMK Faculty,
Russia

Abstract. PARUS is a parallel programing framework that allows you to build parallel programs in data flow graph notation. The data flow graph is created by the developer either manually or automatically with the help of a script. The graph is then converted to C++/MPI source code and linked with PARUS runtime system. The next step is the parallel program execution on a cluster or multiprocessor system. PARUS also implements some approaches for load balancing on heterogeneous multiprocessor system. There is a series of MPI tests that allow developer to estimate the information about communications in a multiprocessor or cluster.

Most commonly, parallel programs are created with the libraries which generate parallel executable code, such as MPI for cluster and distributed memory architectures, and OpenMP for shared memory systems. The tendency to make parallel coding more convenient led to the creation of software front-ends for MPI and OpenMP. These packages are intended to rid the user of part of the problems related to parallel programming. Several examples of such front-ends are DVM [2], Cilk [3], PETSc [4], and PARUS [5], the latter of which is being written by the group of developers headed by the author.

PARUS is intended for writing the program as a data-dependency graph. Writing the program as a data-dependency graph gives the programmer several advantages. In the case of splitting the program into very large parallelly executed blocks it can often be convenient to declare the connections between the block and then execute each block on its own group of processors in the multiprocessor system. The algorithm of the problem being solved is represented as a directed graph where vertices are marked up with series of instructions, and edges correspond to data dependencies. Each edge is directed from the vertex where the data are send from to the vertex which receives the data. It is supposed that the vertex which has received the data will process it some way and then it will send them over to other vertices of the graph. So, the program can be described as a network which has source vertices (which usually serve for reading input files), internal vertices (where the data are processed), and drain vertices, where the data are saved to the output files and the execution terminates. Then the graph is translated into a C++ program which uses the MPI library. The resulting program will automatically try to minimize data trasmission overhead as well as the time of its execution by choosing the processor to execute each vertex of the graph.

One of the targets of this research was to investigate how the data-dependency graph approach to writing parallel programs can be applied to the following examples: a distributed operation over a large array of data, an artificial neural network (perceptron), a frequency filter for sound signals, and multiple alignment problem.

The first test uses a recursive algorithm which computes the result of applying an associative operation to all elements of an array. Two examples of such operations are summation and maximization. Given that every block will be treated by its own processor and the transmission delays are zero, the algorithm will require $O(log_m(n))$ operations, where m is the parameter of the algorithm (data transmission overhead increases with m). Testing this implementation on MVS-1000M with 100 processors exhibited a 40 times speedup on an array sized 10^9.

Next, an algorithm of multiple sequence alignment was implemented. It was a part of the project supported by CRDF grant No. RB01227-MO-2 and by RFBR grant No. 05-07-90238. The program was used to align all human-specific LTR class 5 in the EMBL data bank [1]. The test has demonstrated a 2.4 times speedup on 12 processors on a Prime Power850 machine.

Third, PARUS was used to simulate a three layer perceptron with a maximum of 18,500 neurons in each layer. The maximum acceleration that was achieved was over 7 times.

The fourth test is a solution of the problem which is important in molecular biology. This is a problem of multiple sequence alignment. The parallel implementation was based on the MUSCLE package (http://www.drive5.com/muscle). The procedure of construction of an alignment based on the alignment profiles and cluster sequence tree was parallelized. The data used for testing were all human-specific long terminal repeats (LTR) class 5.

PARUS has been installed and tested on some multiprocessors: MVS1000-M http://www.top500.org/system/5871,
http://www.jscc.ru/cgi-bin/show.cgi?path=/hard/mvs1000m.html&type=3 (cluster 768 Alpha processors), IBM pSeries690 (SMP 16 processors Power4+), Sun Fujitsu PRIMEPOWER 850 Server (SMP 12 processors SPARC64-V).

References

1. Alexeevski A.V., Lukina E.N., Salnikov A.N., Spirin S.A. Database of long terminal repeats in human genome: structure and synchronization with main genome archives //Proceedings of the fours international conference on bioinformatics of genome regulation and structure, Volume 1. BGRS 2004, pp 28-29 Novosibirsk.
2. The DVM system: http://www.keldysh.ru/dvm/
3. The Cilk language: http://supertech.csail.mit.edu/cilk/
4. The PETSc library: http://www-unix.mcs.anl.gov/petsc/petsc-as/
5. The PARUS system: http://parus.sf.net/