Mathematical Problems of Computer Science 23, 2004, 47–53.

On a Software Tool for Implementation of Systolic Algorithms in the Cluster Environment *

Edmon M. Davtyan

Institue for Informatics and Automation Problems of NAS of RA
e-mail edmon@ipia.sci.am

Abstract

In this paper a software tool **SAS** (Systolic Algorithm Simulator) is designed to model the work of a one-dimensional systolic array of n cells on a homogenous computational cluster of $m \ll n$ processors. As a result of the program execution, a cluster-based programming module is obtained, where computational resourses of the system are used in an effective way.

1 Introduction

Let we are given a systolic algorithm [4] to which a one-dimensional systolic array of n cells is assigned. Let we are also given a homogenous computational cluster of $m \ll n$ processors, with pairwise interconnected nodes [4]. Assume that the MPI programming standard [5] and the *mpich* software package environment [7] are used to develop cluster-based parallel programs. Consider the possible ways of implementation of this algorithm, implementation complexities, as well as the advantages and weaknesses that arise out of these (as a qualitative criterion for estimating these advantages and weaknesses, the execution time of the algorithm is considered):

- The work of a systolic array could be realized on one processor using a sequential programming technique. This means that a program could be created which step by step performs the work of the systolic array cells, starting from the first to the last cell. Besides it should be organized in such a way that, depending on the characteristics of the systolic array, a procedure equivalent to data exchange procedure be performed at the end of each step.

- The work of a systolic array could also be realized on one processor using a 'parallel' programming technique, for example with the help of *tread* programming tool. Then a program is created which starts parallel processes in a number equal to the number of array cells and each process performs the work of some cell in the array. In this case also a procedure analogous to data exchange procedure should be organized at the end of each step.

*This research is supported by INTAS - 0447, ISTC - 823 grants and 04.10.31 Target Program of RA.

- The work of the given systolic array could be realized on the base of both one processor and a cluster using MPI standard. In this case usage of *mpich* software package is a possible choice. When choosing this approach neither start of the processes nor the way of organizing data exchange procedure are a matter of trouble for the programmer. The weakness of this choice is that more than one process is performed on a processor, and as a result we have a non-parallel performance of the algorithm and some reduction in algorithm running time.

- An effective way to realize the work of the given systolic array is to partition and distribute the whole work of the systolic array between the cluster nodes. Thus, to find a better solution to this problem, there arises a necessity to construct a new systolic array, the number of cells in which doesn't exceed the number of processors in the cluster. This gives birth to a number of problems: on the length of the array under construction, the choice of local variables in the cells in the new array, definition of interconnections between the cells, organization of computations in the cells, etc. So having the solutions to this problems, one just needs to create a programming module, by using MPI standard, which realizes the new array and runs the programming module on the cluster by executing *mpirun* command [7]. Under such circumstances the usage of an effective algorithm for partitioning the work of the systolic array presented in [1, 2], which remodels the given systolic algorithm, depending on the number of nodes available in the cluster, is a reasonable choice to make.

- To realize the work of the systolic array it is preferable to use a special software tool that receives the given systolic array at its input and automatically generates a cluster-based programming module using the algorithm described in previous item. The user only needs to execute *mpirun* command to run the module, requiring processors of desired number from the ones available in the cluster.

Experiments have shown that if the length of the array is a number of high order, then none of the first three methods described above are effective to solve the problem. In this paper we describe a software tool SAS (Systolic Algorithm Simulator) developed to model the work of a one-dimensional systolic array of n cells on a homogenous computational cluster of $m \ll n$ processors. As a result of the program execution, a cluster-based programming module is obtained, where computational resourses of the system are used in an effective way.

2 Description of the software tool SAS

SAS is a software tool realized under Linux RedHat 9.0 operation system in $mpich - gm$ [8] software package environment, with the use of C++ programming language. Some knowledge of basic concepts of the programming language C, as well as some skills in giving systolic array description components [3] are required to employ the software tool. The description of a systolic array [3] includes array topology, cellular computations and interconnections, problem specification.

In general, the work of SAS is performed in two stages: receipt of the systolic array in some format at the input of the program and construction of the programming module. An application of an effective algorithm for remodeling the systolic array described in [1, 2], modifies and remodels the systolic array received at the input of the program. The

figure below presents both the given systolic array and the systolic array after it has been remodeled:

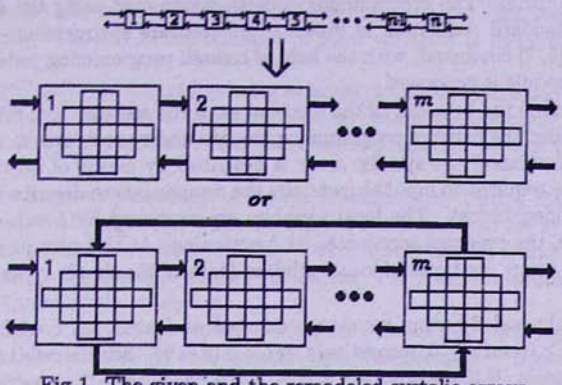

Fig 1. The given and the remodeled systolic arrays

Now we present the list of components of a systolic array description format in SAS:

- *Interconnections between the cells*

  the number of local variables of array cells;

  the number of left–sided output canals (except for the first cell) of array cells;

  interconnections between local variables and left-sided output canals;

  interconnections between local variables and the input canals corresponding to left-sided output canals;

  the number of right-sided output canals (except for the last cell) of array cells;

  interconnections between local variables and the right-sided output canals;

  interconnections between local variables and the input canals corresponding to right-sided output canals;

- *Array structure and cellular computations*

  external data required for performing the work of the systolic array;

  variables that may take new values at each execution of the algorithm;

  the number of array cells;

  the number of steps to perform the work of the systolic array;

  cellular computations;

  initialization function performed before the start of the systolic array work;

  output data entered into the first cell during the work of the array;

  output data entered into the last cell during the work of the array;

  outputs during the work of the systolic array.

Upon receiving the required components in above-mentioned format, the main (kernel) part of software tool SAS generates a programming code in C++ language, which realizes the systolic algorithm. The programming code is constructed using the functions of MPI programming standard realization in *mpich – gm* software environment. Finally, by executing *mpiCC* [5, 7] command, with the help of created programming code, a cluster-based programming module is generated.

Now we describe the behavior of the whole work of the software tool SAS. First of all, it is required to input the name of programming module under construction. At the first stage of program performance, the systolic array is described by means of queries in interactive mode, where it is required to input sequentially the components to describe the systolic array in above-mentioned format. The local variables are expressed by $Loc\_1, \ldots, Loc\_v$, where $v \geq 1$. If $v = 0$, the program terminates its functioning. At this part of the program it is also required to input the types of local variables (each of the simple types in programming language C).

Left- and right- sided output canals are denoted as $Lout\_1, \ldots, Lout\_l$ and $Rout\_1, \ldots, Rout\_r$, where $l \geq 0$ and $r \geq 0$, respectively. If $l = 0$ ($r = 0$), then the cells have no left (right) -sided output canals. If the condition $l = 0 \wedge r = 0$ holds, then the program terminates its functioning. Depending on l, right-sided input canals are denoted as $Rin\_1, \ldots, Rin\_l$, and depending on r, left-sided input canals are denoted as $Lin\_1, \ldots, Lin\_r$.

The external data required for functioning of the systolic array are given by nonempty text file or files. Data files are expressed by variables $Data\_File\_1, \ldots, Data\_File\_f$, where $f \geq 0$. If $f = 0$, then no external data are needed for functioning of the array. In this part of the program, for the file variables it is also required to input physical addresses of corresponding files. For $1 \leq i \leq f$, the variable $Size\_Data\_File\_i$ defines the length of the file $Data\_File\_i$. The function $Datum\_File(Data\_File\_i, index)$ returns the $index$-th element of the file $Data\_File\_i$. Observe that $index \geq 0$ and if $Size\_Data\_File\_i = 0$, then the file $Data\_File\_i$ is empty. The mentioned variables and the function are created by the program.

If to describe the systolic array, variables are required, which can take new values, each time the algorithm is executed, then the variables $Arg\_1, \ldots, Arg\_a$, where $a \geq 0$ are used to express such variables. In this part of the program, it is also required to input the types of variables ($int, char, string$). If $a = 0$, then no such variables are used in the description of the array, otherwise the user himself assigns values to those variables as program parameters at the start of programming module.

The length of the systolic array is defined by the variable *Length*. Each cell is characterized by the variable *cell*, where $cell = 0, \ldots, Length - 1$. The number of algorithm steps is defined by the variable *Steps_Number* and each step is characterized by the variable *step*, where $step = 0, \ldots, Steps\_Number - 1$. This information is shown to the user during the performance of the program SAS.

To set values to the variables *Length* and *Steps_Number* SAS requires to assign a correctly-constructed expression written in either of programming languages C or C++ to each of these variables. Variables $Size\_Data\_File\_1, \ldots, Size\_Data\_File\_f, Arg\_1, \ldots, Arg\_a$ may also be included in the expression.

At the next step it is required to describe the bodies of several functions using the programming codes written in either of programming languages C or C++. All variables and the function mentioned above can participate in those programming codes. Syntactic analysis of input programming codes and above-mentioned expressions has been carried out

with the help of *mpiCC* command.

Below we'll give the list of names of functions that need to be described:

Computations() - the work of each cell in the systolic array at each step is described;

Initialization() - the way of initialization of local variables in the systolic array is described;

InputToFirstCell() - the way of inputting external data in the first array cell is described;

InputToLastCell() - the way of inputting external data in the last array cell is described;

OutputFromArray() - outputs to standard *Output* during the work of the systolic array is described.

This completes the description stage of the systolic algorithm. During the entry procedure the user is allowed to edit and correct the description components of the systolic algorithm. Errors of various types are detected at this stage and in case errors are found, a corresponding message is sent and it is offered to reenter valid data. The user can terminate the work of the program by inputting the symbol q.

The second stage of the software tool is performed without any participation of the user. The whole input data is classified and recorded in a temporary file. Based on the contest of this file, the functions mentioned above, the variables *Length* and *Steps_Number* are declared and defined in programming language C++, as well as the programming code describing the systolic algorithm at the input of the program is constructed also using the same programming language. Naturally, the variables *Length*, *Steps_Number* and the functions *Computations*(), *Initialization*(), *InputToFirstCell*(), *InputToLastCell*(), *OutputFromArray*() which can differ for various systolic arrays are used in the programming code.

As a result of SAS execution, in user's domain of the operating system Linux, a folder having the same name with the programming module is created, which contains the programming module and the *readme.txt* file. In the mentioned text file, the user may find some useful information on the programming module, on running of cluster-based programming module, as well as on input data of the program. The software tool SAS has gone testing in the computational cluster environment ArmCluster.

3 Experimental results

Window-accumulated subsequence matching (WASM) problem has been considered [6]. The problem is: given sequences $t_1, t_2, \ldots, t_n$ and $p_1, p_2, \ldots, p_k$, and a natural number ω, to compute the number of ω windows, where p is a subsequence of t.

The one-dimensional systolic array that solves this problem consists of k cells and performs $n + k$ steps. The array was implemented by SAS and tested on 16 processors of the computational cluster ArmCluster. For the fixed n, k and the fixed number of processors ten tests were performed. The minimal result was chosen. For $n = 10^7$ and $k = 10^3$ the obtained results are given in the diagram in Fig. 2.

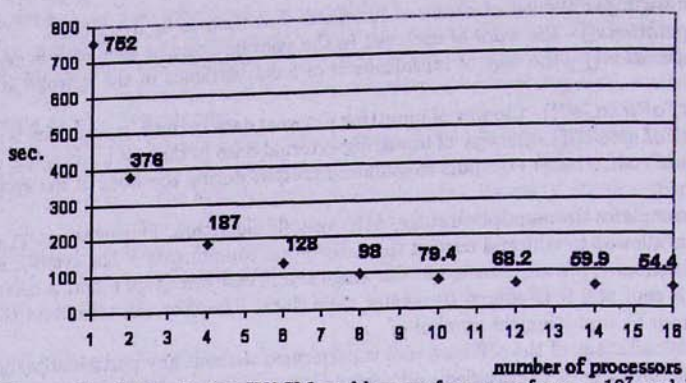

Fig 2. The results obtained after WASM problem performance for $n = 10^7$ and $k = 10^3$

For $n = 10^7$ and $k = 10^4$ the obtained results are given in the diagram in Fig. 3.

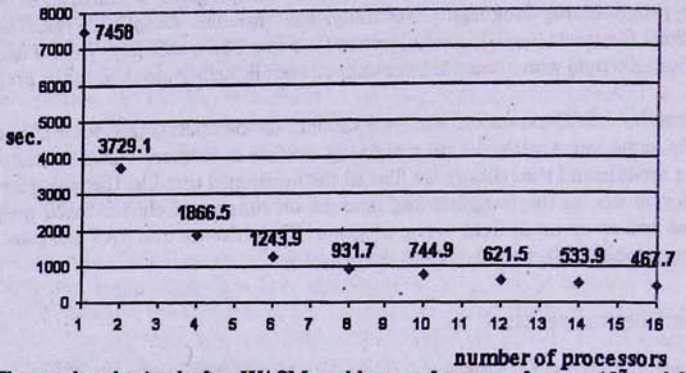

Fig 3. The results obtained after WASM problem performance for $n = 10^7$ and $k = 10^4$

Based on the diagrams the algorithm effectivity was estimated:

$$E_p = \frac{t_1}{t_p \cdot p} \approx 1,$$

where t_1 is the execution time of the algorithm on one processor, t_p is the execution time of the algorithm on p processors.

Acknowledgments. The author gratefully acknowledges helpful discussions and insights by Prof. Yuri Shoukourian. Special thanks to Armen Kocharyan (IIAP) for providing experimental results.

References

[1] E.Davtyan. On the Modelling of One Class of Systolic Structures on a PC Cluster. In proceedings of CSIT-2003, pp. 340-344.

[2] Edmon M. Davtyan. On the Construction of Cluster Systolic Arrays. TRANSACTIONS of IIAP NAS RA,Yerevan, 2004 (in this issue).

[3] Parosh Abdula. Decidable and Undecidable Problems in Systolic Circuit Verification. ACM International Workshop on Formal VLSI Design, Miami, Florida, January 1991.

[4] Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - СПб.: БХВ - Петербург, 2002. - 608 с.: ISBN 5-94157-160-7.

[5] Корнеев В. Параллельное программирование в MPI. Москва-Ижевск: Институт компьютерных исследований, 2003, 203 с.

[6] L. Boasson, P. Cegielski, I. Guessarian, Yu. Matiyasevich. Window-Accumulated Subsequence Matching Problem is Linear. CSIT Conference 2001, Yerevan, Armenia, September 17-20, pp. 74-87.

[7] mpich : http://www-unix.mcs.anl.gov/mpi/mpich

[8] mpich-gm : http://www.myri.com/scs

Կլաստերի միջավայրում սիստոլիկ ալգորիթմների իրականացման ծրագրային գործիքային միջոցի մասին

Է. Մ. Դավթյան

Ամփոփում

Մշակված է SAS (Systolic Algorithm Simulator) ծրագրային գործիքային միջոցը, որը մոդելավորում է n երկարություն ունեցող միաչափ սիստոլիկ զանգվածի աշխատանքը $m \ll n$ պրոցեսորներից բաղկացած համասեռ հաշվողական կլաստերի վրա: Ծրագրի աշխատանքի արդյունքում ստացվում է կլաստերի վրա աշխատող ծրագրային մոդուլ, որը էֆեկտիվ կերպով է օգտագործում համակարգի հաշվողական ռեսուրսները: