Annales UMCS Informatica AI IX, 1 (2009) 27–34
DOI: 10.2478/v10065-009-0003-2

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Parallelization methodology and analysis of benefits from the TRQR program with parallel computing

Marcin Brzuszek[1], Anna Sasak[1], Marcin Turek[2]

[1] *Institute of Computer Science, Maria Curie Sklodowska University, pl. M. Curie-Skłodowskiej 1, 20-031 Lublin, Poland*
[2] *Institute of Physics, Maria Curie Sklodowska University, pl. M. Curie-Skłodowskiej 1, 20-031 Lublin, Poland*

Abstract

Benefits from parallelization of sequential TRQR application were analyzed and presented in this paper. The TRQR program based on the Particle–In–Cell method is used in simulations of particles trajectories in the electromagnetic field. To estimate the performance of parallelized program some formal and informal measures were provided. Analysis was based on three main issues. First - if and how parallel computing influenced program efficiency. Secondly, the way that system reliability increases and thirdly, how this solution improves the extent to which the available hardware computing power is used. The paper also presents the methodology that was used for code re-engineering in order to get a parallel version on the basis of its sequential version.

1. Introduction

There exist plenty of programs that simulate phenomena in different scientific fields. Those programs, designed to work on one-processor machines, were often developed for many years. They were designed to model and simulate reality, which was often functionally limited by computing power of available hardware. A lot of them due to their specifics are and for a long time will be sequential.

On the other hand, some of them are flexible enough to be parallelized. A good example of those programs is TRQR. There are defined many standards that allow programmers to migrate their applications between different computing environments in a relatively easy way. Yet, there exist a lot of aspects that no homogeneous standards were defined for. One of them is the problem of parallelizing the existing sequential code which was not planned to be a basis for future parallel version. The substantial issue is then the choice of proper technique which will be applied in the parallelization process. The chosen technique should not only be efficient but also provide freedom of further code modifications. The TRQR parallelization process is based on methodology presented in paper [**1**].

2. TRQR description

TRQR is based on the Particle–In–Cell (PIC) method that simulates moves of charge particles in the electromagnetic field. The program calculates particles density distribution for chosen parts of ion sources. It also calculates potential distribution in the ion source, analyses particles behaviour in the electromagnetic field and describes the process of beams from the source extraction. It is planned to use such kinds of sources that produce beams of negative ions which, after neutralization, could be used to warm up plasma in the built thermonuclear reactor ITER. Fig. 1 presents the TRQR program scheme.

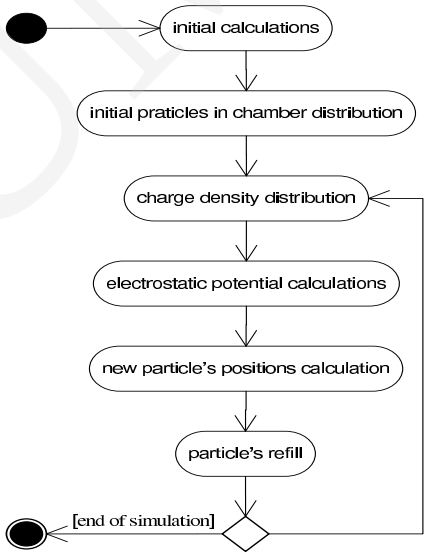

Fig. 1. Simplified activity diagram of the TRQR program

This is a traditional example of simulation of a large number of bodies (so called N–Body problem) [**2**]. In this case, "body" means a macro–particle, representing a lot of real particles of the same kind (electrons or ions). In spite of the PIC assumption that one does not have to follow all real particles, the program has to calculate trajectories for even hundreds of millions of macro–particles and it requires immense time on a single–processor system. To improve reliability of the simulated phenomenon, the program should be provided with some additional functions such as: particles collisions regard, ionization when surface is being hit regard, neutral particles in plasma regard, which would cause the necessity of providing more computing power.

3. Parallelization of sequential code process

The parallelization process was based on the methodology presented in paper [**1**] and illustrated in Fig. 2. It describes the process of gaining an effective, parallel version of the existing sequential program.

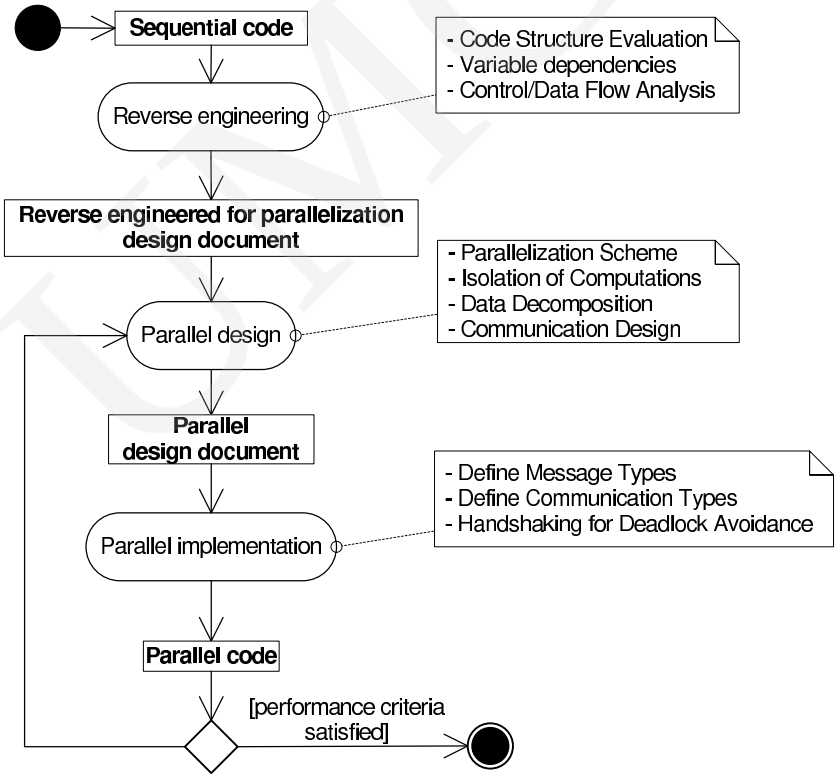

Fig. 2. Methodology for parallelizing sequential code diagram [**1**]

The first step in re-engineering is a reverse engineering stage. This step should make full code structure, variables dependencies and control and data flow analysis. In practice, this step is an accurate analysis and source code reconstruction, whose aim is to understand the programs structure. As a result of the first step, a reverse engineered for parallelization design document is being created. That study gives basis for further parallelization process. Next steps of the adopted methodology are program parallel design and implementation. Designing includes: creating parallelization scheme, data division, isolation of calculation and communication scheme. Implementation means choosing proper parallel middle-ware standard, defining communication method, message types and other calculation coordination related aspects. Designing and implementation are cyclic processes which are repeated until user's performance criteria will be fulfilled. In the case when there is not one algorithm but full program, as a criterion, depending on users' requirements, many different measures can be adopted. In the case of TRQR program parallel design and implementation still remain open. It applies for choosing pieces of code that will be parallelized, choosing problems decomposition method and selecting parallel computation standards. In particular, the choice should be made between implementation for computers with shared or distributed memory.

4. Program performance upgrade

Direct advantage of program parallelization is more effective time use which relates to time assigned to simulation process and therefore improves program performance. Many different elements influence this process in a more or less direct way. For the TRQR program they can be categorized as follows:

1. possibility of running many copies of the same program for different data sets in parallel on separate computers (or separate processors). Due to complexity of a simulated physical phenomenon, it is required to fix value of many program parameters that describe different physical values. Assigning those values influences process accuracy. There must be carry out some shorter and longer simulations to fix them,
2. increasing program execution speed for those input data which have been calculated so far - it is assumed that there exist such data sets that give acceptable results in acceptable simulation time,
3. possibility of gaining a more accurate result by using different input data sets or by fixing other program parameters – it is assumed that there exist such input data sets for which more accurate results are obtained but the simulation time they need makes this simulation useless,

4. possibility of solving a wider problem by adding new, more time consuming functions – it is assumed that program is a functional tool, although due to complexity of the simulated process, it contains numerous generalizations and simplifications and the implemented model does not cover all physical phenomena,
5. possibility of making simulation simultaneously with the real–time visualization process – until the process of visualization is based on partial results saved in files. In all those aspects simulation time influences program performance.

5. Increasing stage of available hardware use

Another benefit from parallelizing a sequential version is increasing the stage that available hardware is being used. Unfortunately, still only great research and development centers own computers with significant computing power. Those centers assign great funds for development of parallel computing systems. Access to those centers computing power almost always requires certain payment for every minute of processor work. Additional limitations are queues, which especially with test simulations could be nagging. The functions which those centers perform and the hardware they offer are undeniable. On the other hand, due to development of available hardware, a remarkable solution should be having parallel computing system of one's own. After appearance of multicore PCs (and the range of this phenomenon), it is obvious that the usage of computing power of available hardware will be increasing. It makes us reconsider programming aspects and search new solutions for this new upcoming reality [**3**].

As for accomplishment of parallelism different presumptions could be made. First, it can be expected that technology development will go in such a direction that all fine–grained parallel work will be moved to hardware. Secondly, some work can be transferred to compiler and then it can be expected on that this compiler will transform the code in such a way that our program will make impression, at least partially, of parallel application. Unfortunately, apart from a few cases consisting of simple data structures, application created as sequential even with the best implemented compilers is not able to give expected performance. There is of course one more way left which means parallel application achievement by programmer. The best situation is when a program is being made from scratch with the adopted guideline to be parallel. In the case when the expanded and optimized sequential version already exists, there could occur some problems with adopting this program to parallel environment. That is why certain methodology is required that would simplify this process.

Programmer must make many decisions: what tools, standards, mechanism to use, whether all program or maybe only the most time consuming fragments should be parallel. One way or another, in the case of existing sequential applications it requires code re-engineering with previous work cost and benefits analysis.

6. System reliability improvement

In the case of many parallel system usage, much more important than performance is high reliability. It is mainly connected with the necessity of stable and trouble-free work. In the case of such scientific programs as TRQR it is not necessary. It is important though, that the hardware used for simulations is available whenever needed. It is especially important for test simulations, performed to fix proper parameters, very often only for certain program pieces. The own multicore PC, in spite of limited computing power, provides some extent of independence. Not to mention that on such computer, in contrary to available multicore machines or clusters (for example local cluster from the Clusterix system) there can be placed numerous tools to analyze program performance, examine processors load and make some real-time visualizations. Well known and popular standards such as MPI (standard based on passing messages between computational nodes) or OpenMP (standard for programming computers with shared memory) were adopted in the TRQR parallelization which gives availability to easy code migrations between different computing environments. It also allows to execute a program on different computers with division for more or less demanding simulations as far as time and number of particles are concerned.

7. Performance criteria

Fundamental performance criterion that was adopted while the TRQR program parallelization is a speedup factor that is described by the formula: $S(p) = \frac{T(1)}{T(p)}$, and efficiency $E(p) = \frac{T(1)}{T(p) \cdot p} = \frac{S(p)}{p}$ where p stands for a number of processors, $T(1)$ and $T(p)$ – the simulation time on one or p processors (adequately) [**2**]. Next to this formal criterion, during the whole process, a very important part was taken by a formal but also very relative measure which is simulation time. During designing and test simulations, it is often decided that working on better (in means of acceleration) solution is not cost–effective enough as predictable benefits in contrary to required development time are not significant enough. Before starting the sequential version parallelization process, there has to be done some costs analysis and some balance has to be adopted between how much time is planned for creating a stable parallel code version and how

much work should be dedicated to its efficiency. The whole time human factor is of great importance here. It is the human who analyses gained results and estimates their usefulness. Yet, still one of indirect factors affecting those decisions is simulation time. Sometimes when the program is being analyzed, it is known in advance that it would take a lot of time. In such a situation it is hard to compare work time with the gained results but still we believe that the experience and conclusion we gain would give a good basis for further studies.

8. Parallel version performance analysis

The direct factor influencing simulation time in the TRQR program is plasma density n, which stands for a number of computing particles in one space unit. The factor n can take values in range $10^9 - 10^{11}$ cm^{-3}. According to the equation $N_{rp} = \frac{n \cdot V_t}{N_m}$, where N_m stands for a number of computing particles in space V_t, plasma density influences the actual number of particles building one computing particle. Obviously, the smaller number of real particles in one computing particle is, the more precise reality picture gained from the results is. Higher plasma density forces usage of more computing particles whose trajectories are being analyzed [**4**].

In the program we managed to parallelize most of critical time pieces, while minimizing communication and synchronization of processes. Part of computing, which we did not get to parallelize, covers less than 10% of all sequential program [**5**]. According to the Amdahl's law we should not get acceleration higher than 10 times [**2**], yet acceleration we managed to acquire exceeds the values resulting from the Amdahl's law. It results from the fact that the size of the problem, which in the case of TRQR means a number of computational particles, can be calibrated according to the processors number. Let us assume that the sequential program N stands for a number of computing units, and $T(1)$ means the simulation time. If the particles number is increased k times, and at the same time the number of processors is increased to k, then simulation time theoretically should come to $T(1)$ as well [**2**]. According to the Gustafson's law every problem that is big enough can be effectively parallelized. The results gained for TRQR match this theory perfectly. In the re–engineering process there were developed the program s versions that give acceptable results. Thus this does not excuse us from further parallelization studies as there exist a number of functionalities which included in the program can force program structures reorganization.

9. Conclusions

The essential and most measurable benefit acquired from the TRQR program parallelization is possibility of significant simulation time reduction. It results from the basic criteria which is acceleration and through–out this it gives plenty of further research possibilities. In the re–engineering process there are created new programs versions. Acceleration values gained for the code parallelization are possible mainly due to problem scalability and deep reverse engineering. Program is not fully functional yet. There exists a certain number of options adding which will require further re–engineering of code. The tool that will be adopted for this process will still be methodology presented in paper [**1**].

References

[1] Andersen P.H., Pizzi J., Zhu R., Cao Y., *Evaluation of a Methodology for the Reverse Engineering and Parallelization of Sequential Code*, PDSE 1999: 124-133.

[2] Wilkinson B., Allen M., Parallel programming, *Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall, 1999.

[3] Gorder P.F., *Multicore Processors for Science and Engineering*, IEEE Computing in Science and Engineering, 2007.

[4] Brzuszek M., Turek M., Sielanko J., *Parallelization of the "Particle-in-Cell" (PIC) density calculations in plasma computer simulations*, IBIZA, 2006.

[5] Turek M., Brzuszek M., Sielanko J., *An MPI-based parallel code for high performance 3–D particle–in–cell ion source plasma simulation*, IBIZA, 2007.