Petro STAKHIV[1], Iryna STRUBYTSKA[2], Yuriy KOZAK[3]

Technical University of Łódz (1), Ternopil National Economic University (2), Lviv Polytechnic National University (3)

Parallelization of calculations using GPU in optimization approach for macromodels construction

Abstract. *Construction of mathematical models for nonlinear dynamical systems using optimization requires significant computation efforts to solve the optimization task. The most CPU time is required by optimization procedure for goal function calculations, which is repeated many times for different model parameters. This allows to use processors with SIMD architecture of calculation parallelization. The effectiveness of such parallelization is the subject of investigation in this paper.*

Streszczenie. *Rozwiązywanie problemów optymalizacyjnych dla nieliniowych układów dynamicznych wymaga dużych nakładów obliczeniowych. Większość czasu procesora pochłaniane jest przez obliczanie wartości funkcji celu, co powtarzane jest wielokrotnie dla różnych parametrów modelu. Dzięki temu możliwe jest wykorzystanie architektury SIMD do zrównoleglenia obliczeń. Przedmiotem przedstawionych badań jest efektywność takiego zrównoleglenia. (**Zrównoleglenie obliczeń przy pomocy GPU dla problemów tworzenia makromodeli**)*

Keywords: macromodels, optimization, parallelization, GPU, SIMD.
Słowa kluczowe: makromodele, optymalizacja, zrównoleglenie, GPU, SIMD

Problem Statement

The process of design and analysis of modern dynamical systems with large number of components of different nature requires significant computation resources. It is caused by considerable number of components of the system being designed and variety of physical phenomena to be taken into account. The usage of macromodels in such conditions allows significant reducing of required computation efforts because it makes it possible to ignore not important effects for particular analysis. Macromodels can be used to describe single components as well as subsystems of significant size including elements of different nature. Such state of the problem to be considered leads to the necessity to develop universal approaches intended for construction of complex dynamical object macromodels in the form useful for their further analysis

There are many approaches and methods which might be used to construct macromodels of nonlinear dynamical systems but their usage is limited due to complexity of the problem. Therefore the task to develop an universal approach for nonlinear dynamical macromodels construction is still not solved.

One promising approach, which has enough universality, is the use of optimization. But this approach has a significant disadvantage: the optimization task to be solved in this approach is very complex and often requires too big computation efforts for solution. So techniques for simplification of this optimization task are being developed [1].

One more direction which allows to reduce the time needed to solve optimization task using parallelization based on processors with SIMD architecture is considered in this paper.

Processors with SIMD architecture have much better ratio performance/price then regular computer processors. Authors used graphical processor NVIDIA GeForce GTS250, 1024 Mb which has SIMD architecture to verify the approach.

The usage of optimization for construction of macromodels of nonlinear dynamical systems

Let's consider some system, which need to be simulated (see fig. 1). Here $\vec{x}$ - variables which describe system state, $\vec{v}$ - variables which describe external influences, and $\vec{y}$ - variables which describe output values.

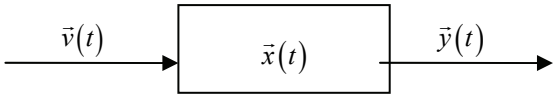

Fig.1 System to be simulated

Due to we are considering dynamical systems all three mentioned vectors are functions of time. This can be either continuous functions or sets of discrete values.

Not limiting the generality let's concentrate on analysis of discrete values case, because discrete values is more useful for computers. We will use the discrete state variables form:

$$(1) \qquad \begin{cases} \vec{\mathbf{x}}^{(k+1)} = \mathbf{F} \cdot \vec{\mathbf{x}}^{(k)} + \mathbf{G} \cdot \vec{\mathbf{v}}^{(k)} + \mathbf{\Phi}(\vec{\mathbf{x}}^{(k)}, \vec{\mathbf{v}}^{(k)}) \\ \vec{\mathbf{y}}^{(k+1)} = \mathbf{C} \cdot \vec{\mathbf{x}}^{(k+1)} + \mathbf{D} \cdot \vec{\mathbf{v}}^{(k)} \end{cases},$$

where F, G, C, D – some matrixes with unknown coefficients, which need to be found during model construction, $\mathbf{\Phi}(\vec{\mathbf{x}}^{(k)}, \vec{\mathbf{v}}^{(k)})$ - some vector-function, the form and coefficients of which should also be found.

The mathematical form (1) we selected for our system simulation as well as any other form, has some set of unknown coefficients $\vec{\lambda}$. In our case it includes elements of matrixes F, G, C, D and coefficients of vector-function $\mathbf{\Phi}(\vec{\mathbf{x}}^{(k)}, \vec{\mathbf{v}}^{(k)})$.

Let's also introduce some criterion for model precision measuring $Q(\vec{\lambda}) > 0$, which depicts an inaccuracy of system simulation using the model for known time intervals.

Function $Q(\vec{\lambda})$, also called a goal function is generally calculated as a root-mean-square error of system simulation using constructed model.

$$(2) \qquad Q\left(\vec{\lambda}\right) = \sum_i \sum_k \left(\vec{\tilde{y}}_i^{(k)} - \vec{y}_i^{(k)} \right)^2,$$

where $\vec{\tilde{y}}_i^{(k)}$ is the response of the object, calculation using macromodel, $\vec{y}$ is the response of the object measured at experiment.

Therefore model construction can be reduced to finding a value of vector $\vec{\lambda}$, for which function Q will reach it's minimum. This is done using optimization algorithm.

The task of finding a valley point of nonlinear function Q is a complex task, which is considered in many writings. Our goal function is hard to calculate in terms of needed CPU time for one calculation. The time needed for single calculation of goal function depends linearly from the number of samples, because the goal function is a sum of deviations between real values and values simulated using constructed model for all samples. For construction of a high-quality model the number of samples used for goal function calculation will be significant, thus CPU time needed for single calculation of goal function will also be significant.

The time needed for single calculation of goal function proportionally depends on the number of points in transient characteristics used for its calculation. The number of goal function calculations needed to find its minimum significantly depends on the number of unknown coefficients which need to be found. For most optimization algorithms this dependency is exponential. These two facts together results in practical problems for the usage of optimization approach for construction of macromodels of nonlinear dynamical objects, which are described by many coefficients. Currently there are papers presenting techniques for simplification of optimization task for example by splitting the model construction process into a set of stages [1].

Therefore, because practical usage of the considered approach requires a lot of CPU power, it is worthwhile to consider using parallelization. Taking into account, that goal function calculation for different values of vector $\vec{\lambda}$ is independent and is done using same algorithm we can state, that processors with SIMD architecture can be effectively used for this task. Such processors allow to execute same instructions for many data sets. SIMD systems includes many identical processor units executing same instruction sets but using independent data sets. Most common device, build on SIMD architecture is GPU (Graphics Processing Unit). Modern graphical processors are multy-processor and multy-core systems with SIMD architecture which have high productivity (up to 1 TFLOPs). They have better ratio "price/performance" then traditional architectures (for example computation clasters). So GPU can be effectively used not only for processing of graphics but also for execution of other computational tasks [4].

Mentioned approach can be used together with most optimization algorithms. The requirements for optimization algorithm are:

1. algorithm should require goal function to be calculated many times at one iteration. The effectiveness of parallelization, as it will be shown below, grows with increase of the number of goal function calculations at one iteration
2. the set of coefficients $\vec{\lambda}$ for all points where goal function is calculated at one iteration should not depend on result of goal function calculation in other points at same iteration.

An example of the algorithm, which satisfies these requirements, is Rastrigin's director cone method. An example of algorithm, which can not be used with considered parallelization, is Nelder–Mead method. Also there are algorithms for which mentioned criteria are satisfied partially. For such algorithms effective usage of considered parallelization technique requires switching between parallel and sequential calculation of goal function during optimization process. An example of such algorithm is Gradient descent method, which does not satisfy second criterion because after determination of gradient direction the algorithm requires goal function to be calculated one more time, and position of the point where it should be calculated depends on results of goal function calculations already conducted at the same iteration.

For estimation of the effectiveness of the considered parallelization approach authors implemented it for calculation of goal function (2) and model form (1).

The procedure of parallel calculation of goal function was implemented using CUDA (Compute Unified Device Architecture) technology.

NVIDIA CUDA technology is a fundamentally new technology for conducting calculations using graphical processors. It is designed to handle business and technical calculations. CUDA technology allows to write programs on standard C programming language easily which can use productivity of new generation NVIDIA graphical processors executing thousands of threads and process big amounts of data. This technology is very useful for computation tasks which allow parallelization. It works on a variety of NVIDIA graphical processors, and significantly simplifies writing programs with utilization of computational power of GPU cards.

With CUDA programming model the computation task can be first split into a set off parts (blocks), which can be executed in parallel. Then each block is split into a set of threads, which is executed in parallel and can depend one from another. A block In general is a sub-task, which is executed independently [3].

In our case we need to perform an identical set of operations to calculate a goal function for different model coefficients. So it is enough to have just one block and many threads. All calculation threads use same transient characteristics, so to minimize data transfer between main CPU and GPU we need to load them into GPU once at the very beginning.

In general, the optimization algorithm with the usage of GPU for goal function calculation will perform next steps (see fig. 2).

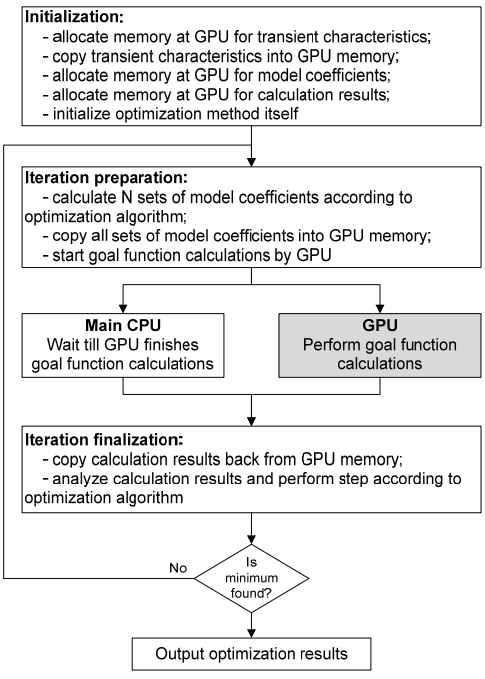

Fig.2 the optimization algorithm with the usage of GPU for goal function calculation

With CUDA programming model it is not a complex task to implement this parallelization approach. We will skip implementation details here as CUDA documentation at http://www.nvidia.com contains enough information to implement such algorithm.

Proposed approach for calculation parallelization can be used with different optimization algorithms. Therefore authors did not take the computation time needed for the algorithm itself into consideration. At practice the computation time for the algorithm itself is negligibly lower then computation time needed for calculations of the goal function. Thus results archived by authors will be correct for the majority of cases.

For estimation of the effectiveness of proposed parallelization approach we will compare time needed for goal function calculations with the usage of parallelization and without it.

$$S = \frac{t_c}{t_p},$$

where S is archived speed-up level, t_c is time needed for sequential calculation using main CPU, t_p is timeneededfor parallel calculation using GPU.

It is obvious, that the performance of the main CPU is much higher then the performance of the single core of GPU. So we expect that parallel implementation will be more effective when goal function is calculated for many sets of model coefficients simultaneously. The dependency of archived speed-up level from the number of parallel goal function calculations are shown at fig. 3 (average value in seconds for 100 experiments).

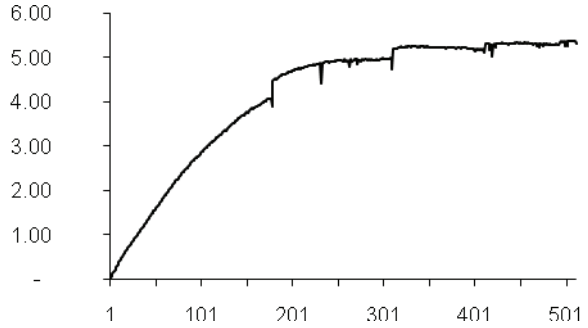

Fig.3 Effectiveness of parallelization using GPU (NVIDIA GeForce GTS250, 1024 Mb) in comparision with Main CPU (Core2Duo E8400, 3GHz).

As we can see, for large numbers of parallel goal function calculation threads we were able to reach up to 5 times speed-up in comparison with the main CPU even though we used one of the simplest GPU cards with CUDA support.

For sequential calculations performance (measured in operations per second) does not depend on the number of goal function calculations. So dependency of performance from the number of goal function calculations for parallel implementation will have exactly same dependency as shown at fig. 3.

It is also interesting to compare time, needed for all goal function calculations from the number of goal function calculations. It is obvious that for sequential calculation this dependency is linear. For parallel calculations this dependency is shown at fig 4.

This dependency shows that we have only small increase in time needed to perform all goal function calculations with increasing number of such calculations. This additionally proves that parallelization effectiveness is increasing with increase of the number of goal function calculations at one iteration of optimization algorithm.

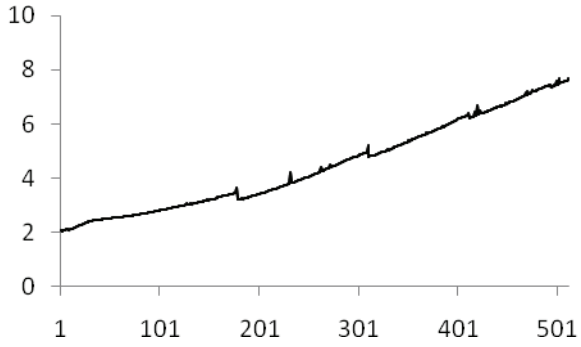

Fig. 4 time needed to calculate goal function for all needed coefficient sets (in ms) as a function of the number of coefficient sets

Additional limitations are present for calculation precision. Modern GPUs can execute calculations with double precision, but calculation speed is significantly reduced in this case. Graphical processors are progressing rapidly, so this problem should be solved in the near future.

Conclusion

Proposed approach for calculation parallelization in optimization approach for macromodels construction was successfully implemented and tested using NVIDIA GeForce GTS250, 1024 Mb graphical processor. The effectiveness of proposed approach depends on the number of goal function calculations performed on one iteration of optimization algorithm.

Authors were able to reach about 5 time speed-up when using GPU in comparison with main CPU (Core2Duo E8400, 3GHz) when more then 350 coefficient sets were tested at one iteration. This result is interpreted as good by authors because the comparison was done between relatively slow GPU and relatively new main CPU.

REFERENCES

[1] P. Stakhiv, Yu. Kozak, Design of discrete models of complex dynamic systems based on optimization approach, Proceedings of the 11th International Conference on “System modeling and Control”, Zakopane, Poland, October 17-19, 2005, pp. 297-301

[2] Liliana Bychkovska-Lipinska L., Stakhiv P., Kozak Y., Parallelization of Calculations in Construction of Mathematical Models using Optimization, Proceedings of XIII International Conference on “System modeling and Control”, SMC’2009, ISBN 978-83-927875-0-1

[3] NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, Version 2.0, – 2008. –107 p.

[4] А. В. Боресков, А. А. Харламов. Основы работы с технологией CUDA. – М.: ДМК Пресс, 2010. – 232 с.

***Authors**: Prof. Petro Stakhiv - Institute of Computer Sciences, Technical University of Łódz, ul. Wolczanska 215, Łódz, 90-924, Poland, petro.stakhiv@p.lodz.pl; Iryna Strubytska – Department of Computer Scince, Institute of Computer Informational Technology, Ternopil National Economic University, Yunosti Str. 9, 46020 Ternopil, Ukraine, iryna.str@gmail.com; Assoc. Prof. Yuriy Kozak - Department of Theoretical Electrical Engineering, Institute of Electric Power and Control Systems, Lviv Polytechnic National University, 12 S. Bandera str, Lviv, 79013, Ukraine ykozak@mail.ru*