Programs and Algorithms of Numerical Matematics 16
J. Chleboun, K. Segeth, J. Šístek, T. Vejchodský (Eds.)
Institute of Mathematics AS CR, Prague 2013

MASSIVE PARALLEL IMPLEMENTATION OF ODE SOLVERS

Cyril Fischer

[1] Institute of Theoretical and Applied Mechanics AS CR, v.v.i.
Prosecká 76, Prague 9, Czech Republic
fischerc@itam.cas.cz

Abstract

The presented contribution maps the possibilities of exploitation of the massive parallel computational hardware (namely GPU) for solution of the initial value problems of ordinary differential equations. Two cases are discussed: parallel solution of a single ODE and parallel execution of scalar ODE solvers. Whereas the advantages of the special architecture in the case of a single ODE are problematic, repeated solution of a single ODE for different data can profit from the parallel architecture. However, special algorithms have to be used even in the latter case to avoid code divergence between individual computational threads. The topic is illustrated on several examples.

1. Introduction

The modern Graphical Processor Unit (GPU) serves as a powerful graphics engine thanks to its highly parallel programmable processor. As a parallel device it features peak arithmetic and memory bandwidth that substantially outpaces its CPU counterpart. Contemporary graphics processors thus operate as co-processors within the host computer.

There are several peculiarities in the GPU architecture from the point of view of a regular PC user, namely rather complicated memory access and parallel architecture of the graphics multiprocessor. Whereas a high level programming interface can unify the memory access up to certain limit, parallel algorithms for GPUs have to be treated in a special way, see e.g. [6].

Graphics processors are built as multithreaded SIMD (single instruction, multiple data) devices. It means that each instruction of the code is preformed on a set of data at once. Any exception in data treatment (like data dependent branches) results in splitting of the program flow and leads to a significant degradation of performance.

2. Parallel solution of an ordinary differential equation

Solution of an ordinary differential equation is a sequential problem in its nature. There are only several parallelisable points in a general procedure of an ODE solver, usually regarding evaluation of the integrated functions.

In his exhaustive review paper [2] Burrage follows the classification of the seminal Gear's work [4]. He divides the techniques into two different categories: parallelism across the method and parallelism across the system. The first group comprises methods which exploit concurrent function evaluations within one ore more steps. The second group includes methods based on waveform relaxation. These methods decouple the original system into a set of small and independent subsystems which can then be solved in parallel.

According to literature survey, both categories are still being developed, but not in conjunction with GPU computation. The high cost of communication between CPU and GPU and demand of synchronous operation on the whole set of data make this computational environment specific. To achieve improvement if a GPU is used, any of the methods mentioned above ought to request thousands of function evaluation for each step. On the other hand, if the dimension of the coupled ODE system is large, a single RHS evaluation can exploit GPU accelerated matrix/vector multiplication, matrix inverse evaluation etc.

Only a few papers dealing with ODEs and GPU are availalabe up to now. They mostly reflect problems originating from biomechanics or chemistry, see e.g. [10] or are dealing with evolutionary differential equations. Only three projects can be found on internet which aim at implementation ODE solvers to the GPU hardware:

CULSODA [3] is an adaptation of the highly sophisticated algorithm LSODA (adaptive steplength, automatic stiff/non-stiff method switching etc, see [5]) for NVIDIA's CUDA compiler. It does not contain any parallel code in itself. The procedure can be used to perform a parameter study of a single ODE system. As it will be shown in the next section, usability of the CULSODA code is rather limited. It seems that the project has been abandoned since 2009.

ODEINT_V2 [1] is a general C++ library for numerical solution of ODE. With most integration methods it offers exploitation of CUDA capable hardware. The authors claim that the GPU is worth to employ in the following applications: parameter studies, large systems like ensembles of lattices, discretizations of PDEs, etc.

CUDA-sim [9, 10] is a Python package providing CUDA GPU-accelerated biochemical network simulation. The package offers ODE solver based on CULSODA implementation and stochastic differential equation solver according to the Euler-Maruyama algorithm. It's usage is limited to the case when a large number of independent ODEs are to be solved en bloc.

It seems that none of the sophisticated methods mentioned in [2] or newer papers is implemented in the available libraries. On the other hand, there are certain tasks which are based on solution of a large number of independent ODEs or systems. Even if the beautiful theory of parallel ODE solvers cannot be employed in this cases, existing cheap GPUs could significantly speed up the computation in such a situation as it will be shown in the next section.

3. Example

Two tasks routinely performed by engineers are good candidates for a parallel processing example: computation of *response spectra* and *resonance curve.*

Response spectra $\rho(\omega)$ of recorded time history is a plot of the peak or steady-state response of a series of oscillators of varying natural frequency ω^2 that are forced into motion by the recorded signal $a(t)$, cf. (1). The natural frequency of the oscillators is taken as the independent variable, coefficient of damping β is predefined as a parameter, see e.g. [8]. Resonance curve $R(\omega)$ is a similar plot of the peak or steady-state response of a structure described by a system of differential equations $f(t)$, that is forced into motion by a harmonic function $a_0 \sin(\omega t)$. In this case the independent variable is the frequency ω of the harmonic input motion (2) .

$$\rho(\omega) = \max_{0<t<T} |y(t)| , \qquad \text{where } \ddot{y} + 2\beta\dot{y} + \omega^2 y = -\ddot{a} \tag{1}$$

$$R(\omega) = \max_{0<t<T} |y(t)| , \qquad \text{where } f(y, \dot{y}, \ddot{y}, \ldots, t) = a_0 \sin(\omega t) \tag{2}$$

The both problems are similar and lead to solution of a large number of independent initial value problems. In this case, the parallel computation is an obvious choice.

Evaluation of the resonance spectra will be illustrated using the following example. The equation (3) describes movement of the mathematical pendulum with an external excitation in the suspension point (see the detailed derivation in [7]):

$$\left.\begin{aligned} \ddot{\xi} + \frac{1}{2r^2}\xi\frac{\mathsf{d}^2}{\mathsf{d}t^2}(\xi^2+\zeta^2) + 2\beta_\xi\dot{\xi} + \omega_0^2\left(\xi + \frac{1}{2r^2}\xi(\xi^2+\zeta^2)\right) &= -\ddot{a} \qquad (a) \\ \ddot{\zeta} + \frac{1}{2r^2}\zeta\frac{\mathsf{d}^2}{\mathsf{d}t^2}(\xi^2+\zeta^2) + 2\beta_\zeta\dot{\zeta} + \omega_0^2\left(\zeta + \frac{1}{2r^2}\zeta(\xi^2+\zeta^2)\right) &= 0 \qquad (b) \end{aligned}\right\} \tag{3}$$

where ξ, ζ are components of the projection of the pendulum's bob to the (xy) plane, r is the length of the pendulum, $\omega_0^2 = g/r$ is the natural frequency of the corresponding linear pendulum and g is the gravitational acceleration. The viscous damping is denoted as β_ξ, β_ζ in respective directions. As the harmonic excitation $a(t) = a_0 \sin(\omega t)$ acts in the ξ direction only, the basic type of motion takes course in the vertical (xz) plane if the time history starts under homogeneous initial conditions. With the increasing amplitude of the excitation $a(t)$, the in-plane movement can lose its stability and movement in the transversal direction ζ can occur.

If the resonance curve is to be computed, it is necessary to perform numerical solution of the system (3) for a large number of excitation frequencies ω. Two numerical methods enter into comparison: the CULSODA solver and in-house implemented simple backward Euler solver.

Table 1 shows the timings for CULSODA solver and various numbers of ODE solution threads. Two experiments are shown: the real case, when each thread solves ODE for its own excitation frequency, and the unrealistic one when all threads do exactly the same work (the frequency ω is the same for all the threads). The first case shows the devastative effect of the thread divergence on the overall performance

# values	1024	8192	16 384	32 768	1024	8192	16 384	32 768
	thread divergence ($\omega = 1, ..., 10$)				no thread divergence ($\omega = 1$)			
GPU (sec)	19.2	40.7	70.7	127.3	0.4	0.9	1.7	3.4
CPU (sec)	0.54	4.21	8.54	16.4	0.4	2.8	6.0	11.1

Table 1: Timing of the resonance curve enumeration using CULSODA in single prec. CPU: 2× Intel Xeon X5560 (16 threads), GPU: NVIDIA Quadro 4000 (256 cores).

# values	64	1024	8192	16 384	32 768
GPU (sec)	1.75	1.93	2.97	5.84	11.06
CPU (sec)	0.74	8.89	72.49	143.21	295.1

Table 2: Timing of the resonance curve enumeration using simple backward Euler method with constant step length. CPU: 2× Intel Xeon X5560 (16 threads, single precision), GPU: NVIDIA Quadro 4000 (256 cores, single precision).

of the GPU. The second case shows the theoretical potential of the GPU. Starting from certain problem size (1000 threads in this example) GPU is processing faster than CPU. The thread divergence in the first case is caused by two reasons: (a) the equation significantly changes its properties for growing ω and (b) the LSODA code changes its course accordingly.

The adaptivity of the LSODA algorithm is a great disadvantage when used in the GPU code. To eliminate the thread divergence a simple backward Euler ODE solver has been implemented. To avoid any unnecessary jumps and conditions, the linear equation solver procedure used in the Newton method has been hard-coded for a predetermined dimension using the Crammer rule. Number of iterations of the Newton method has been fixed. The method exhibits reasonable accuracy for a sufficiently small step. Table 2 lists the corresponding timings for the backward Euler solver. There is no doubt about the winner in this case: starting with 256 computational threads the GPU becomes significantly faster.

The both results are not intended to compare the individual methods. In a scalar case the LSODE algorithm is faster. In order to keep the accuracy reasonable the backward Euler code uses much smaller step size than the LSODE solver. Moreover, whereas the LSODE requires about 2 function evaluation for each step and one Jacobian for (about) 10 steps, the backward Euler evaluates $f(y, t)$ and the Jacobian in each step and for each iteration of the Newton method.

The response spectrum is usually computed using the Newmark method. The Newmark method is an explicit second order method whose recurrence is tailored for the second order equation of motion, see (1). It is especially convenient in the (usual) case when some measured record enters the computation as a set of regularly sampled discrete values. It supposes a fixed length of the integration step. Figure 1 shows timing of the implementation for different numbers of frequencies. As the simple

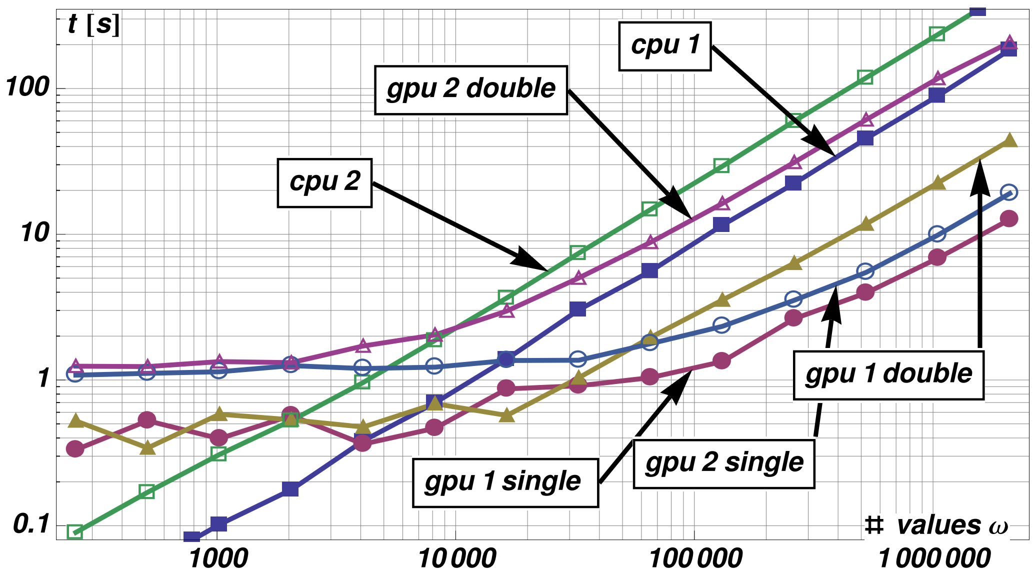

Figure 1: Timing of response spectra enumeration (El Centro earthquake record, 31 000 samples, $\Delta t = 0.001$) using the Newmark recurrence formula. The CPU timing for single and double precision does not differ.
CPU 1: 2× Intel Xeon X5560 (16 threads), GPU 1: NVIDIA Quadro 4000 (256 cores). CPU 2: Intel i7 950 (8 threads), GPU 2 : NVIDIA GT430 (64 cores)

Newmark recurrence does not contain any branches, thread divergence cannot occur in this case.

The speed difference between single and double precision arithmetic is shown in Figure 1. The performance penalty for double precision is 1/2 for the advanced NVIDIA Quadro 4000 GPU and 1/8 for the entry level GT430. The ratios correspond well to the architectures of the both cards (one double precision floating point unit is common for 2 cores in the high end NVIDIA GPUs or for 8 cores in GT430). However, even the slower card can compete well with the dual Xeon workstation for well chosen problems.

4. Conclusions

The great progress of the computer technology enables the scientific community to solve fairly complex problems. Current GPUs are cheap devices with power of a supercomputer. This demands the scientists to adopt higher level of knowledge of programming techniques.

We have shown two examples of GPU utilization for numerical solution of initial value problems. Three numerical methods were presented: i) The adaptive solver LSODA provided to be the least advantageous for GPU. ii) As an alternative, the backward Euler method is reasonably accurate and stable and can be programmed in a manner which meet requirements of a fast GPU code. iii) Simple recurrence formula, simple arithmetic, no special functions, no jumps or branches and no inter-thread communications make the Newmark procedure an ideal candidate for employment of GPU.

It seems that the advanced methods for solution of a single large ODE system are not very convenient when used for machines with SIMD architecture. However, the presented examples exhibited possibility of fruitful utilization GPUs in practice. It has been shown that in certain cases even an entry-level GPU can work faster than a high-end CPU.

Acknowledgements

The kind support of the Czech Scientific Foundation No. 103/09/0094, Grant Agency of the ASCR No. A200710902 and RVO 68378297 institutional support are gratefully acknowledged.

References

[1] Ahnert, K. and Mulanski, M.: ODEINT_v2, `http://www.odent.com/`, 2012

[2] Burrage, K.: Parallel methods for initial value problems. Applied Numerical Mathematics. **11** (1993), 5–25.

[3] CULSODA, `http://code.google.com/p/culsoda/`, 2009

[4] Gear, C. W.: Parallel methods for ordinary differential equations. Calcolo **25** (1988), 1–20.

[5] Hindmarsh, A. C.: ODEPACK, A Systematized Collection of ODE Solvers, Scientific Computing. In Stepleman,R. S. et al. (Eds.) *IMACS Transactions on Scientific Computation*, vol. 1. , pp. 55–64. Amsterdam, North-Holland, 1983.

[6] Kirk, D. B. and Hwu, W. W.: *Programming Massively Parallel Processors: A Hands-on Approach.* Morgan Kaufmann Publishers, Burlington, 2010.

[7] Náprstek, J. and Fischer, C.: Auto-parametric semi-trivial and post-critical response of a spherical pendulum damper. Comp. Struct. **87** (2009). 1204–1215.

[8] Newmark, N. M. and Hall, W. J.: Earthquake Spectra and Design, *Engineering Monographs on Earthquake Criteria, Str uctural Design, and Strong Motion Records*, vol. 3. Earthquake Eng. Res. Inst., Oakland, California, 1982.

[9] Zhou, Y. and Barnes, C.: CUDA-sim, `http://www.theosysbio.bio.ic.ac.uk/ /resources/cuda-sim/`, Version 0.07, 21/12/2011

[10] Zhou, Y., Liepe, J., Sheng, X., Stumpf, M. P. H. and Barnes, C.: GPU accelerated biochemical network simulation. Bioinformatics **27** (2011), 874–876.