Stochastic Analysis of a Queue Length Model Using a Graphics Processing Unit

J. Přikryl*
Faculty of Transportation Sciences, Czech University of Technology, Prague, Czech Republic
Institute of Information Theory and Automation, Czech Academy of Sciences, Prague, Czech Republic
** Corresponding author: prikryl@utia.cas.cz*

J. Kocijan
Jožef Stefan Institute, Ljubljana, Slovenia
Centre for Systems and Information Technologies, University of Nova Gorica, Slovenia

DOI: 10.2478/v10158-012-0007-2

ABSTRACT: Mathematical modeling is an inevitable part of system analysis and design in science and engineering. When a parametric mathematical description is used, the issue of the parameter estimation accuracy arises. Models with uncertain parameter values can be evaluated using various methods and computer simulation is among the most popular in the engineering community. Nevertheless, an exhaustive numerical analysis of models with numerous uncertain parameters requires a substantial computational effort. The purpose of this paper is to show how the computation can be accelerated using a parallel configuration of graphics processing units (GPU). The assessment of the computational speedup is illustrated with a case study. The case study is a simulation of Highway Capacity Manual 2000 Queue Model with selected uncertain parameters. The computational results show that the parallel computation solution is efficient for a larger amount of samples when the initial and communication overhead of parallel computation becomes a sufficiently small part of the whole process.

KEY WORDS: Graphics processing unit, GPU, Monte Carlo simulation, computer simulation, modeling.

1 INTRODUCTION

Computer simulation is recognized as one of the most frequently used tools in system analysis and design in science and engineering. Technology development enables increasingly sophisticated mathematical models to be simulated in less and less time. On the other hand it is also true that the increasing capability of hardware and software does not prevent scientists and engineers to reach the computational limits of the hardware.

The purpose of this paper is to show how computation with a personal computer can be accelerated using a parallel configuration of graphics processing units (GPU). This is done from the user's point-of-view to test the usability of the compared computational platforms for simulation.

The change of computational configuration is validated and illustrated with the case of a Monte Carlo simulation of a dynamic system model from the field of transportation

science. Other such applications are increasingly popular, e.g., Raina et al. (2009), Catanzaro et al. (2008).

Mathematical models are inevitable in science and engineering. When parametric mathematical description is used, the issue of parameter estimation accuracy arises. Models with uncertain parameter values can be evaluated with various methods and computer simulation is among the most popular in the engineering community. Nevertheless, an exhaustive numerical analysis of models with numerous uncertain parameters requires a substantial computational effort. An example of such an exhaustive numerical analysis is the Monte Carlo simulation of dynamic systems with uncertain parameters, see e.g., Ray and Stengel (1993), Calafiore and Dabbene (2006).

The structure of the paper is as follows: the next section describes the hardware used and the implementation of simulation software for graphics processing units. Section 3 describes the selected case study and the comparison of computational time for selected hardware configurations. Conclusions are stated at the end.

2 COMPUTATIONAL ACCELERATION WITH GPU

While “standard” CPUs continue to provide users with more and more computing power nowadays (Shen and Lipasti, 2005), many computer scientists migrate towards general-processing GPU (GPGPU) applications (Kirk and Hwu, 2010), using graphics card processors as parallel accelerators for memory-dense, floating-point intensive applications. GPGPU accelerators are becoming the tool of choice in many computationally-bound research tasks such as bioinformatics or numerical modeling and simulation, and also in traffic simulation (Strippgen and Nagel, 2009).

The concept of a GPGPU processor evolved from the needs of 3D-graphics-intensive applications. This need dictated the design of the processor in such a way that more transistors were dedicated to the data processing rather than to control and data caching, as in a regular CPU. Subsequently, the processor was designed to be able to execute a data-parallel algorithm on a stream of data; consequently, the GPGPU processors are sometimes called “stream processors”. The currently dominant architectures for GPGPU computing are the nVidia CUDA (nVidia, 2011) and the AMD APP (formerly ATI Stream) (AMD, 2011). Graphics processing units are currently a low-cost, high performance computing alternative. With their intrinsic parallel structure they allow for significant computational increase in speed in comparison to the single processor architecture.

In order to show how the computation problem of Monte Carlo numerical analysis of a traffic model using the Matlab package for numerical computation (MathWorks, 2010) can benefit from the use of currently available GPGPU hardware, we describe the computing architectures that were used in our tests. The basic characteristic of the tested hardware configurations is given in Table 1.

For demonstration purposes, two different hardware configurations and two different software configurations will be used:

- Multiple-core CPU (CPU). A standard PC was used, equipped with a four-core Intel i5/750 processor with 4MB of cache memory (1MB per core). The Matlab interpreter will use its built-in multiple-core capabilities, resulting in most of the elementary matrix operations being computed in parallel on all four processor cores.

- Multiple-core GPU (GPU). The same PC platform as in the tests above will be used, but the most computationally intensive parts of the code, namely the Monte-Carlo simulation of the HCM model, will be offloaded to the GPU. We will use the NVIDIA

GeForce GTX 275 GPU, which includes 30 streaming multiprocessors with 8 cores each (in total 240 processor cores). Every processor may use up to 16 kB of fast shared memory similar to the cache memory of a traditional CPU.

- The first software configuration will use the GPU-accelerated Jacket library for Matlab (AccelerEyes, 2011). The library allows for almost seamless conversion of an existing Matlab code into a code that runs on a GPU.

- In order to assess the efficiency of the Jacket library, we will also use manually programmed GPU code in the form of a MEX file (an external routine, directly callable from the Matlab code).

We have written a small benchmark program based on Matlab, which tests the execution times of a typical HCM simulation cycle. The program gathers computation times in relation to the number of samples used in the simulation. This approach provides us with a view on the impact that different architectures have on the computation time from the user's perspective.

Table 1: Parameters of our hardware configurations: the used GPU is the first-generation GPU with double precision support and, as such, its double precision performance is 8 times lower than that of a single precision computation.

	Multiple-core CPU	*Multiple-core GPU*
Type	i5/750	GTX275
Cores	4	30×8 (240)
Cache memory	8MB	240×16kB (~4MB)
Total memory	4096MB	896MB
Memory bandwidth	17 GB/s	127 GB/s
GFLOPS (float)	42.56	1010
GFLOPS (double)	42.56	124

The NVIDIA GPGPUs come with the CUDA API (Garland et al., 2008), which is used to directly program the GPU hardware. While Jacket makes the use of CUDA (and other third party libraries) internally, for the assessment of Jacket efficiency we had to implement two custom GPU programs (kernels).

Although it is relatively easy to manually setup and perform basic mathematical operations on a GPU, it quickly becomes more complex when dealing with more demanding numerical problems. Additionally, due to the GPGPU architecture, special care must be taken when performing memory operations:

- due to relatively slow memory transfer, data transfers between the host system and the GPU device shall be as few as possible, and shall be asynchronous if possible,

- improper kernel code design, with respect to the operation on different memory types (main GPU memory, shared memory, constant memory, texture memory) and ignoring memory access coalescing on the GPU device, can cause a significant performance loss,

- shared memory in a block is organized into banks and accessing elements not consecutively will cause a bank conflict,

- shared memory and processor register space is scarce, and care should be taken to limit the number of kernel variables to as low as possible; this issue is critical, even more with double precision arithmetic, as a double precision number occupies two register units.

3 CASE STUDY

As an example of a mathematical model that will be used for Monte Carlo analysis the Highway Capacity Manual 2000 Queue Model for back of queue presented in the Highway Capacity Manual (2000) will be considered. The model is composed of two queue components,

$$Q_{k+1} = Q_{1,k+1} + Q_{2,k+1}, \tag{1}$$

where $Q_{1,k+1}$ represents an average back of queue for uniform arrival distribution that is corrected by a multiplicative correction factor accounting for queue progression, and $Q_{2,k+1}$ is an additive correction term accounting for randomness and uncertainty in the queue development process.

Under the assumption that the cycle length C is equal to the period of data collection T_Δ, the original model can be reformulated as

$$Q_{1,k+1} = \mathrm{PF}_{2,k} \frac{\frac{v_{\mathrm{L},k} C}{3600}(1 - z_k)}{1 - \min(1.0, X_{\mathrm{L},k}) \cdot z_k}, \tag{2}$$

where $\mathrm{PF}_{2,k}$ is an multiplicative correction factor adjusting the queue length for the effects of progression, $z_k = g_k/C$ is the relative green signal length in the k-th cycle corresponding to the absolute green length g_k, and $X_{\mathrm{L},k} = v_{\mathrm{L},k}/c_{\mathrm{L},k}$ is the lane saturation ratio computed from the current traffic volume $v_{\mathrm{L},k}$ and the lane capacity $c_{\mathrm{L},k}$. The correction factor is given by

$$\mathrm{PF}_{2,k} = \frac{(1 - R_{\mathrm{P},k} \cdot z_k)(1 - y_k)}{(1 - z_k)(1 - R_{\mathrm{P},k} \cdot y_k)},$$

where

$$y_k = \frac{v_{\mathrm{L},k}}{s_{\mathrm{L},k}}$$

is a flow ratio of the approach expressing its degree of total saturation for the current saturation flow $s_{\mathrm{L},k}$ and

$$R_{\mathrm{P},k} = \frac{P}{z_k}$$

is the platoon ratio of the approach concerned. Here, the variable P specifies the proportion of vehicles arriving during the green signal.

The second term of (1) attempts to correct possible errors in (2), caused by uncertain and random factors, such as the previous queue length, or a non-uniform arrival distribution. Assuming approximately constant flows during the data collection period T_Δ it is defined as

$$Q_{2,k+1} = \frac{c_{\mathrm{L},k} \cdot T_\Delta}{4}\left(X_{\mathrm{L},k} - 1 + \sqrt{(X_{\mathrm{L},k} - 1)^2 + \frac{8 k_{\mathrm{B},k}\, x_k}{c_{\mathrm{L},k} \cdot T_\Delta} + \frac{16\, k_{\mathrm{B},k}\, Q_k}{(c_{\mathrm{L},k} \cdot T_\Delta)^2}} \right). \tag{3}$$

The second-term adjustment factor $k_{\mathrm{B},k}$ accounts for early arrivals and for actuated signals and is defined as

$$k_{\mathrm{B},k} = 0.01 I_k \left(\frac{s_{\mathrm{L},k} \cdot g_k}{3600} \right)^{0.6}. \tag{4}$$

The upstream filtering factor I_k in this equation expresses the influence of saturation ratio $X_{\mathrm{u},k}$ at the upstream intersection on platoon arrivals at the modeled intersection. In our case it is computed using formula given by the Highway Capacity Manual (2000) as

$$I_k = 1 - X_{\mathrm{u},k}^{2.68}. \tag{5}$$

After obtaining the average back of queue by evaluating (1), a percentile back of queue factor has to be applied to the predicted value to get a more conservative prediction of the queue length. The percentile correction factor $f_{\mathrm{B}\%}$ is defined as

$$f_{\mathrm{B}\%}(Q) = p_1 + p_2 \cdot e^{-\frac{Q}{p_3}} \tag{6}$$

where parameters p_1, p_2 and p_3 are usually determined by the desired percentile (Highway Capacity Manual, 2000). The final queue length $Q_{\%}$ incorporating the percentile correction is then

$$Q_{\%,k+1} = Q_{k+1} \cdot f_{\mathrm{B}\%}(Q_{k+1}). \tag{7}$$

In our case parameters p_1, p_2 and p_3 are uncertain parameters for which we have only interval values. These values are $p_1 \in [1.1, 1.5]$, $p_2 \in [0.2, 0.4]$, $p_3 \in [25, 35]$.

The Monte Carlo simulation with 10^6 simulation runs where parameter values were uniformly distributed within their intervals was run within Matlab on both previously mentioned configurations. Simulation results are depicted in Figure 1.

3.1 User effort

As we would like to compare the results not only in terms of acceleration, but also in the context of user effort that went into particular variants of the tested software, we will now summarize the latter.

- Multiple-core computer (CPU). In the case of a recent Matlab version, the Matlab computing kernel automatically uses a multithreaded version of certain functions and operations in case that the size of operands exceeds a certain limit.

- Personal computer with GPU using Jacket (Jacket). The transition from pure Matlab to "Jacketized" code has been very swift. The original Matlab function that represents the model had to undergo just minor changes and has been compiled by Jacket as a kernel code. The other change was changing the model simulation to run element-by-element (rather than vectorized) in a loop using a special GPU optimized loop construct provided by Jacket. The total effort was a few hours for a person fairly familiar with the Matlab environment.

- Personal computer with manually programmed GPU (GPU). The simulation procedure has been rewritten as a MEX file, manually implementing the model as a GPU kernel. In addition, as Jacket provides accelerated implementations of functions for finding mean value, maxima and minima of a data vector, these three operations have been implemented as custom kernels too. The current code snapshot represents a few days of programming and debugging effort for an experienced programmer. Most of the effort probably went into debugging the model code due to its high register space requirements.

3.2 Computational effort

The comparison of computational times is illustrated with a case study of a Monte Carlo simulation using from 100000 to 6400000 samples of a HCM model with changing parameters p_1, p_2 and p_3. The computational effort assessment is given in Figure 2.

We can see that for a low number of samples the overhead of GPU computation severely affects the computation – the time needed to set up the GPU hardware and to transfer the data from the host computer to the GPU severely affects the performance of the GPU-accelerated code.

Once reaching above approximately a hundred thousand samples, the full advantage of the GPU capabilities starts to be visible: The "Jacketized" code, despite the minimum effort necessary for the changes of the original HCM model, reaches a speedup of a factor of more then 15, and the manually programmed version of the HCM model outperforms the CPU version by factor of more than 25.

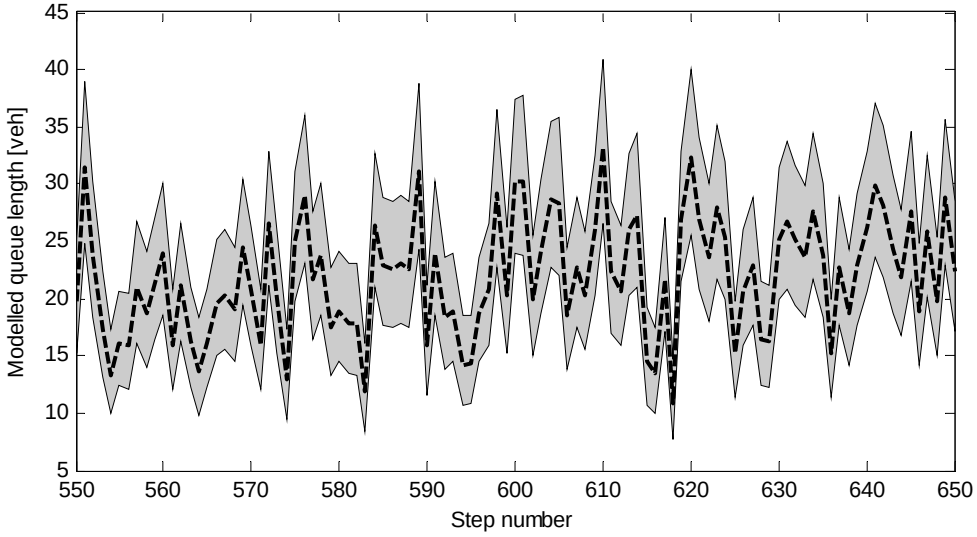

Figure 1: Response of the queue-length model with stochastic parameters p_1, p_2 and p_3.

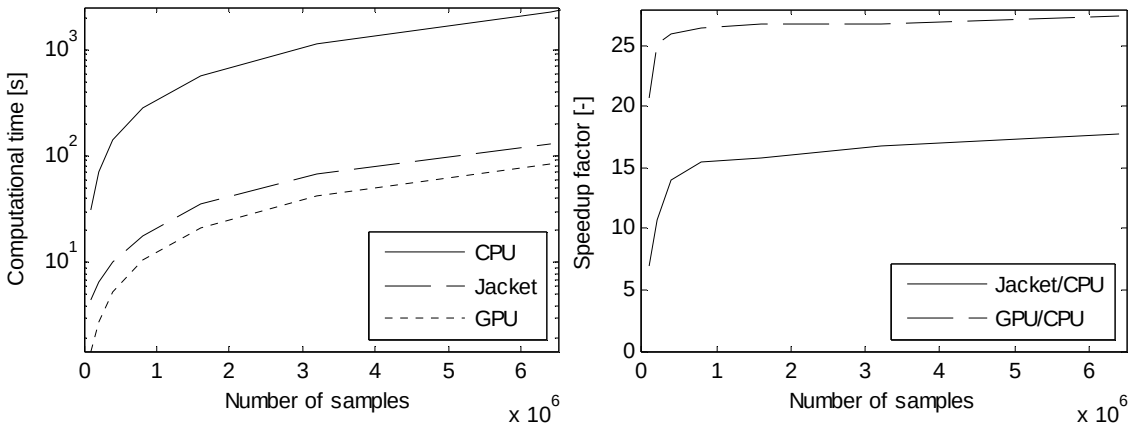

Figure 2: Computation times of the model Monte Carlo simulation versus the number of simulation runs for different hardware configurations (left). Relative speedups of a simulation run on a personal computer utilizing graphics processors, with respect to the multi-core computation (right).

4 CONCLUSIONS

Computer simulation is a flexible and frequent tool that can be used for analysis and design in science and engineering. When the amount of simulation runs is increased – as it is the case of the Monte Carlo simulation – and the models are complex, the drawback is an increasing computational time. This paper provides a description of the implementation of the Monte Carlo simulation on graphics processor units and a comparison of computational-time with a standard multi-core personal computer on a dynamic system simulation case-study. The assessment was performed from the user's point-of-view to test the usability of the compared computational platforms for simulation.

The assessment of the simulation algorithm implementation on nVidia GTX275 graphics processing unit for the Highway Capacity Manual 2000 Queue Model revealed that even a straightforward acceleration using a third party library for GPU computation (Jacket) can increase the simulation speed by a factor of more than 15 and that a speedup of more than 25 can be reached by manually programming the GPU hardware. It has to be noted that the GPU used is currently a middle-class device and that its computing capabilities in double precision floating point arithmetic are inferior to state-of-the-art devices.

As hardware capabilities are improving constantly and research on efficient algorithms is on-going the presented assessment might not be of long-term value. However, it offers a state-of-the-art comparison of an affordable hardware configuration that might help to circumvent the computational issue in intermediate time before more efficient algorithms or better technologies arise. With this it fulfills the purpose for which it was intended.

ACKNOWLEDGMENTS

This work has been supported by the Slovenian Research Agency, grants Nos. P2-0001 and J2-2099, and by a bilateral project between Slovenia and Czech Republic 'System Identification Based on Gaussian Process Model for Traffic Control Applications' (MEB091015).

REFERENCES

AccelerEyes, 2011. *Jacket User Guide* Version 1.8.1. Atlanta, GA (USA): AccelerEyes.

AMD, 2011. *AMD Accelerated Parallel Processing OpenCL Programming Guide.* Sunnyvale, CA (USA): Advanced Micro Devices, Inc.

Calafiore, G., Dabbene, F. (Eds.), 2006. *Probabilistic and Randomized Methods for Design under Uncertainty*. London: Springer. ISBN 978-1846280948.

Catanzaro, B., Sundaram, N., Keutzer, K., 2008. Fast support vector machine training and classification on graphics processors. In *Proceedings of 25th Annual International conference on Machine Learning*. New York: Omnipress, pp. 104 – 111.

Garland, M., Grand, S. L., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., Volkov, V., 2008. Parallel computing experiences with CUDA. *Micro, IEEE*, vol. 28, no. 4, pp. 13 – 27. ISSN 0272-1732.

Highway Capacity Manual 2000. Washington (USA): Transportation Research Board, Federal Highway Administration.

Matlab 2010b, 2010. Natick, MA (USA): MathWorks Ltd.

NVIDIA**,** 2011. *CUDA Programming Guide* Version 4.0. Santa Clara, CA: NVIDIA Corporation.

Kirk, D. B., Hwu, W. W., 2010. *Programming Massively Parallel Processors – A Hands-on Approach.* 1st edition. Burlington, MA: Morgan Kaufmann (USA). 280 p. ISBN 978-0123814722.

Raina, R., Madhavan, A., Ng, A. Y., 2009. Largescale deep unsupervised learning using graphics processors. In *Proceedings of International Conference on Machine Learning*.

Ray, L. R., Stengel, R. F., 1993. A Monte Carlo approach to the analysis of control system robustness. *Automatica*, vol. 29, no. 1, pp. 229–236. ISSN 0005-1098.

Shen, J. P., Lipasti, M. H., 2005. *Modern Processor Design: Fundamentals of Superscalar Processors*. McGraw-Hill Series in Electrical and Computer Engineering. New York (USA): McGraw-Hill. 640 p.

Strippgen, D., Nagel, K., 2009. Multi-agent traffic simulation with CUDA. In *Proceedings High Performance Computing & Simulation (HPCS '09)*, pp.106 – 114.

Volkov, V., Demmel, J., May 2008. *LU, QR and Cholesky factorizations using vector capabilities of GPUs.* Technical Report No. UCB/EECS-2008-49 [online]. Berkeley: EECS Department, University of California. Retrieved from: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-49.html.