Improving ODE integration on graphics processing units by reducing thread divergence

Thomas Kovac[1,2], Tom Haber[1], Frank Van Reeth[1], and Niel Hens[2,3]

[1] Expertise centre for Digital Media, Hasselt University, Belgium
[2] Center for Statistics, Hasselt University, Belgium
[3] Chermid, Vaccine and Infectious Disease Institute, University of Antwerp, Belgium
{thomas.kovac,tom.haber,frank.vanreeth}@uhasselt.be
niel.hens@uantwerpen.be

Abstract. Ordinary differential equations are widely used for the mathematical modeling of complex systems in biology and statistics. Since the analysis of such models needs to be performed using numerical integration, many applications can be gravely limited by the computational cost. This paper present a general-purpose integrator that runs massively parallel on graphics processing units. By minimizing thread divergence and bundling similar tasks using linear regression, execution time can be reduced by 40-80% when compared to a naive GPU implementation. Compared to a 36-core CPU implementation, a 150 fold runtime improvement is measured.

Keywords: Pharmacometrics · Epidemiology · Parallelism · High-Performance Computing · Graphics Processing Units

1 Introduction

Systems of coupled ordinary differential equations (ODEs) are widely used for mathematical modeling of complex systems in epidemiology [6, 8], biology [9] and pharmacology [5]. The analysis of such models is usually performed by numerical integration, since analytical solutions can not be derived in general. Additionally, many applications need to carry out a massive amount of simulations for which the computational cost can be a serious limitation.

Performing such analyses on a central processing unit (CPU) can be quite time-consuming, even on today's multi-core processors. Graphics processing units (GPU) have moved from being common computer graphics and image processing instruments to powerful general-purpose devices [12]. GPUs are single-instruction multiple-data (SIMD) devices, consisting out of hundreds of cores, that give access to tera-scale performance on common workstations. The architecture is ideally suited for executing identical and independent operations on different data, such as image processing.

This paper mainly focuses on ODE simulations that can be performed independently and embarrassingly parallel. Many applications already exhibit this behavior or the underlying algorithm can often be changed to be more favorable to this situation. For example: sensitivity and parameter sweep analysis inherently require the evaluation of many parameters. Optimization and Bayesian sampling algorithms are typically sequential in nature, but parallel alternatives exist for both: evolutionary/genetic algorithms [4, 7, 8] and sequential Monte-Carlo methods [3, 11]. Alternatively, computing expectations [15] over differential equation based models is again massively parallel.

While the embarrassingly parallel nature matches perfectly with the GPUs capabilities, branching in all but the simplest integration methods can cause divergent program paths and a significant drop in performance [1]. The proposed method rearranges tasks among available threads such that they are less likely to diverge. A linear regression model is constructed to predict which tasks are similar in behavior and grouped together accordingly.

2 Background

2.1 GPU

NVIDIA GPUs are comprised out of different layers of parallel processing; the top level consists of Streaming Multiprocessors (SMs). Blocks of a kernel are mapped onto an SM, which in turn executes it using user-allocated threads. The multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps. For optimal performance, threads within a warp are required to execute the same instruction at any given time. As a result, whenever two or more threads diverge, operations of both branches need to be executed for all warp threads. This can lead to a serious drop in performance [1].

2.2 Integration Methods

Among numerical integration algorithms, Runge-Kutta methods are a family of explicit iterative methods, with a wide variety of orders and schemes [13]. The Dormand-Prince method [2], also known *DOPRI*, is a fifth-order method where the step-size is adjusted by the truncation error, which is approximated by the difference between the fourth and fifth-order estimates. MATLAB's *ode45* is also an implementation of the DOPRI method.

While these methods cannot cope well with *stiff* ODEs, many statistical or biological models only exhibit stiffness at extreme parameter values.

2.3 Epidemiological Models

Epidemiologists use mathematical modeling of infectious diseases to improve insight into disease dynamics, resulting in the creation of more effective vaccines and antiviral drugs, better intervention/vaccination programs [6, 8, 15].

One epidemiological model is the Susceptible-Exposed-Infected-Recovered (SEIR) model which describes the flow of individuals through these mutually exclusive disease states. This model is extensible to more complex diseases by adding compartments. Santermans et al. [15] studied the Ebola outbreak of 2015 in West Africa. Equation 1 models SEIR dynamics over time, where $S(t)$, $E(t)$, $I(t)$, and $R(t)$ are the number of susceptibles, exposed, infected and recovered, respectively and $N(t) = S(t) + E(t) + I(t) + R(t)$ denotes population size.

$$\begin{cases} \frac{\mathrm{dS(t)}}{\mathrm{dt}} = -\beta(t)S(t)\frac{I(t)}{N(t)}, \\ \frac{\mathrm{dE(t)}}{\mathrm{dt}} = \beta(t)S(t)\frac{I(t)}{N(t)} - \gamma E(t), \\ \frac{\mathrm{dI(t)}}{\mathrm{dt}} = \gamma E(t) - \alpha I(t) - \sigma I(t), \\ \frac{\mathrm{dR(t)}}{\mathrm{dt}} = \sigma I(t), \end{cases} \tag{1}$$

3 Related Work

Seen et al. [16] show that a Runge-Kutta-Fehlberg method (*RK45*) with adaptive step-size on GPU outperforms a CPU implementation, given that the problem dimensions are large enough, as in 200 equations, or more. Having a Runge-Kutta implementation with adaptive time steps, however, suffers from a phenomenon called variable task-length [10]. The step-size modification can vary between individual GPU threads, resulting in warp divergence and loss in performance. Murray et al. [10] suggest bundling multiple data items into each thread, allowing a thread to immediately advance onto the next task once an item is completed.

Stone et al. [18] implemented two parallel strategies; a so-called “one-thread” method and “one-block” method. The former method employs one thread to solve an ODE, the latter uses an entire block of threads. Both *RK45* and CVODE integration methods were ported to GPU using aforementioned parallel strategies. Significant speedups were reported of all GPU adaptations, but without thread divergence even greater speedups are possible.

Stone et al. [17] emphasize that an efficient and effective ODE integrator must employ the available instruction-level parallelism of the underlying hardware as well as the numerical efficiency. Having implemented a non-stiff Runge-Kutta ODE solver on both GPU and Xeon Phi, the authors report the GPU version being slower than the Xeon Phi version as thread divergence caused by the adaptive step-sizes negatively impacts performance.

4 Methods

Related work shows multiple successful Runge-Kutta method ports to GPU. Solutions where each thread solves an ODE suffer from performance loss due to varying length of tasks caused by thread divergence. Results that employ multiple threads to solve an ODE overcome this problem [17, 8], however not nearly as many ODEs can be solved as compared to one-threaded solutions. In this paper, the DOPRI integration method, an NVIDIA Tesla P100 GPU, a one-threaded solution are employed and thread divergence is strongly reduced.

Algorithm 1 Dormand-Prince

```
1: function INTEGRATE(t, tOut)
2:     while t <= tOut do                          ▷ try to make a step with size dt
3:         ynew, error ← TryStep(dt)
4:         if error $\leq$ rtol then                        ▷ step was successful
5:             t ← t + dt
6:             y ← ynew
7:             dt ← GrowStepsize(dt, error)          ▷ potentially grow stepsize
8:         else
9:             dt ← ShrinkStepsize(dt, error)        ▷ step failed, shrink stepsize
       return y
```

As can be seen in Algorithm 1, there are two causes of thread divergence in one-threaded solutions: the test whether or not a step was successful and the number of steps taken during integration. The former is less of a problem since the branches are very small and care was taken to move all common operations out of them. The latter is the main cause of performance loss since it results in threads that idle for a long time. The number of steps required can wildly differ from parameter to parameter. Ensuring that threads performing similar tasks by bundling them by the required number of steps, accomplished by sorting followed by partitioning, automatically reduces thread divergence as all threads within a warp execute the same instructions.

A linear regression model is used to predict the number of integration steps a task will require. This model is trained a-priori on a small set of parameters (1000) and is used at runtime to group tasks that are similar in number of steps. Transforming the rate parameters to log-space results in a higher predictive performance. To demonstrate general applicability, the number of steps is also predicted for the Nimotuzumab model [14]. Figure 1 shows that the model accurately predicts the number of integration steps required, given the parameters, for both the SEIR and Nimotuzumab model.

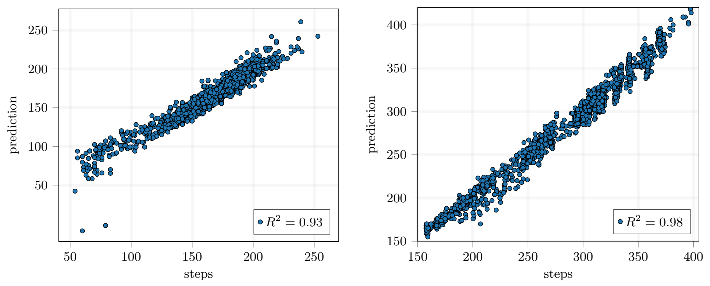

Fig. 1. Number of actual steps plotted against the prediction for the SEIR model (left) and the Nimotuzumab model (right). For both models that the number of integration steps required, based on the parameters, can be accurately predicted.

5 Results

All results are based on the SEIR model from Santermans et al. [15]. 10K parameters were sampled from their prior to create a realistic dataset. Since runtime depends heavily on the parameters, random subsets of 7168 parameters are created and performance distributions are shown. The predictor is always trained on random subsets of 1000 parameters.

The top left plot of Figure 2 shows the distribution of the number of steps attempted by the integrator. While the average number of steps is 160, the spread is quite significant. The top right plot compares distributions of runtime for random subset of parameters, bundled using the predictor as well as bundled employing a-priori knowledge of the number of steps. Bundling tasks according to similarity clearly has a positive effect on performance. On average, a 40% increase is observed and the predictor performs slightly worse compared to the case with a-priori knowledge. Compared to a CPU implementation (36-core Intel Skylake Xeon 6140; 192GB RAM), a 150-fold improvement in runtime is measured.

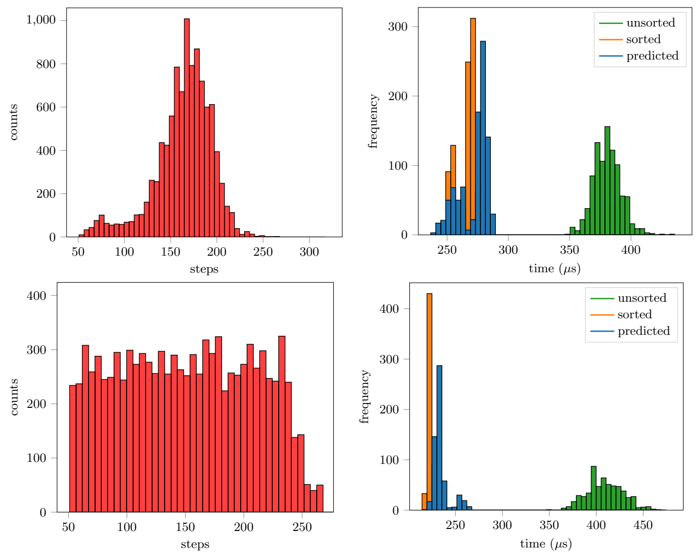

Fig. 2. Left: distributions of integrator steps of the original dataset sampled from the prior [15] and an altered dataset where the parameters were re-sampled such that the distribution is approximately uniform. Right: distribution of runtime for random subsets of the original dataset, bundled using a-priori knowledge of the number of steps and bundled using the predictor for both datasets.

To indicate the impact of the distribution of steps, the original dataset is re-sampled such that the distribution of steps is approximately uniform. Bottom plots of Figure 2 show the performance distribution for the uniform dataset. On average, the performance is increased by 80%.

Clearly, when the distribution of steps is extremely peaked, the performance difference will be quite small. In the worst case, every parameter will require the same number of steps with no change in performance due to bundling.

Figure 3 shows performance for a implementation which forces the integrator to use the same step-size for all threads in a warp similar to Murray et al. [10]. After attempting a step and computing the error, a local shuffle operation is performed to compute the largest error and all threads continue with this error instead. As a result, all threads will take the same branch and make the same decision in terms of step-size. However, due to the additional cost of the local shuffle as well as the extra steps taken by the integrator, this method is roughly 2x slower. Even for this implementation bundling the parameters helps.

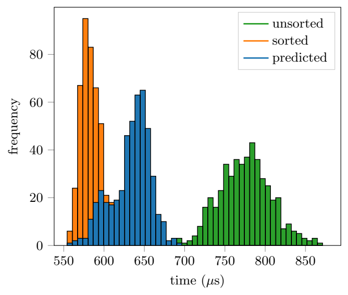

Fig. 3. Runtime distribution for an implementation forcing fixed step-sizes for all threads within the same warp and a comparison with bundling the tasks.

6 Conclusion and Future Work

GPUs are massively parallel single-instruction multiple-data devices capable of tera-scale performance. However, the small differential equation based models typically used in statistics and biology are not ideally suited for these devices due to the branching nature of numerical integration algorithms. This branching causes SIMD units to diverge, resulting in a significant performance drop. By rearranging the order of tasks, bundling similar tasks together, this problem can be alleviated to some extent. A linear regression model, trained on a small set of parameters, is used for predicting the similarity of tasks. The runtime improvement depends on the distribution of the number of steps in the tasks: the more spread out the distribution, the bigger the expected improvement. The experiments show improvements of 40% to 80%.

The global relation between parameters and number of steps can sometimes be hard to capture with a predictive model. However, optimization algorithms and Bayesian samplers typically explore the parameter space only locally. Therefore, future work will look into an online learning method which captures only this local behavior and requires no a-priori training.

References

1. Bialas, P., Strzelecki, A.: Benchmarking the cost of thread divergence in cuda. In: International Conference on Parallel Processing and Applied Mathematics. pp. 570–579. Springer (2015)
2. Dormand, J.R., Prince, P.: A family of embedded Runge-Kutta formulae. Journal of Computational and Applied Mathematics **6**(1), 19–26 (1980)
3. Doucet, A., Freitas, N., Gordon, N. (eds.): Sequential Monte Carlo Methods in Practice. Springer New York (2001)
4. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer Berlin Heidelberg (2003)
5. van der Graaf, P.H., Benson, N.: Systems pharmacology: Bridging systems biology and pharmacokinetics-pharmacodynamics (PKPD) in drug discovery and development. Pharmaceutical Research **28**(7), 1460–1464 (may 2011)
6. Hens, N., Shkedy, Z., Aerts, M., Faes, C., Damme, P., Beutels, P.: Modeling Infectious Disease Parameters Based on Serological and Social Contact Data: A Modern Statistical Perspective, vol. 63. Springer Science & Business Media (03 2013)
7. Kennedy, J., Eberhart, R.: Particle Swarm Optimization (1995)
8. Kovac, T., Haber, T., Van Reeth, F., Hens, N.: Heterogeneous computing for epidemiological model fitting and simulation. BMC Bioinformatics **19**(1), 101 (2018)
9. Murray, J.D.: Mathematical Biology I. An Introduction, Interdisciplinary Applied Mathematics, vol. 17. Springer, 3 edn. (2002)
10. Murray, L.: GPU Acceleration of Runge-Kutta Integrators. IEEE Transactions on Parallel and Distributed Systems **23**(1), 94–101 (2012)
11. Nemeth, B., Haber, T., Liesenborgs, J., Lamotte, W.: Relaxing scalability limits with speculative parallelism in sequential monte carlo. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER). IEEE (sep 2018)
12. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.: A survey of general-purpose computation on graphics hardware (2007)
13. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes 3rd Edition: The Art of Scientific Computing. Cambridge University Press, New York, NY, USA, 3 edn. (2007)
14. Rodríguez-Vera, L., Ramos-Suzarte, M., Fernández-Sánchez, E., Soriano, J.L., Guitart, C.P., Hernández, G.C., Jacobo-Cabral, C.O., de Castro Suárez, N., Codina, H.C.: Semimechanistic model to characterize nonlinear pharmacokinetics of nimotuzumab in patients with advanced breast cancer. The Journal of Clinical Pharmacology **55**(8), 888–898 (2015)
15. Santermans, E., Robesyn, E., Ganyani, T., Sudre, B., Faes, C., Quinten, C., Bortel, W.V., Haber, T., Kovac, T., Reeth, F.V., Testa, M., Hens, N., Plachouras, D.: Spatiotemporal evolution of ebola virus disease at sub-national level during the 2014 west africa epidemic: Model scrutiny and data meagreness. PLOS ONE **11**(1) (jan 2016)
16. Seen, W.M., Gobithaasan, R.U., Miura, K.T., Ismail, M.T., Ahmad, S., Rahman, R.A.: GPU Acceleration of Runge Kutta-Fehlberg and Its Comparison with Dormand-Prince Method. AIP Conference Proceedings **1605**(1), 16–21 (2014)
17. Stone, C.P., Alferman, A.T., Niemeyer, K.E.: Accelerating finite-rate chemical kinetics with coprocessors: Comparing vectorization methods on GPUs, MICs, and CPUs. Computer Physics Communications **226**, 18–29 (2018)
18. Stone, C.P., Davis, R.L.: Techniques for solving stiff chemical kinetics on graphical processing units. Journal of Propulsion and Power **29**(4), 764–773 (Jun 2013)