

METHODOLOGY ARTICLE

Open Access



# Heterogeneous computing for epidemiological model fitting and simulation

Thomas Kovac<sup>1,2\*</sup> , Tom Haber<sup>2†</sup>, Frank Van Reeth<sup>2</sup> and Niel Hens<sup>1,3</sup>

## Abstract

**Background:** Over the last years, substantial effort has been put into enhancing our arsenal in fighting epidemics from both technological and theoretical perspectives with scientists from different fields teaming up for rapid assessment of potentially urgent situations. This paper focusses on the computational aspects of infectious disease models and applies commonly available graphics processing units (GPUs) for the simulation of these models. However, fully utilizing the resources of both CPUs and GPUs requires a carefully balanced heterogeneous approach.

**Results:** The contribution of this paper is twofold. First, an efficient GPU implementation for evaluating a small-scale ODE model; here, the basic S(usceptible)-I(nfected)-R(ecovered) model, is discussed. Second, an asynchronous particle swarm optimization (PSO) implementation is proposed where batches of particles are sent asynchronously from the host (CPU) to the GPU for evaluation. The ultimate goal is to infer model parameters that enable the model to correctly describe observed data. The particles of the PSO algorithm are candidate parameters of the model; finding the right one is a matter of optimizing the likelihood function which quantifies how well the model describes the observed data. By employing a heterogeneous approach, in which both CPU and GPU are kept busy with useful work, speedups of 10 to 12 times can be achieved on a moderate machine with a high-end consumer GPU as compared to a high-end system with 32 CPU cores.

**Conclusions:** Utilizing GPUs for parameter inference can bring considerable increases in performance using average host systems with high-end consumer GPUs. Future studies should evaluate the benefit of using newer CPU and GPU architectures as well as applying this method to more complex epidemiological scenarios.

**Keywords:** ODE, PDE, Infectious diseases, Epidemiology, SIR model, GPU, Asynchronous, Parallel, Particle swarm optimization, Heterogeneous computing

## Background

Over the last years, substantial effort has been put into enhancing our arsenal in fighting epidemics from both technological and theoretical perspectives. For example, effective vaccines and antiviral drugs can be produced with knowledge going deep into the molecular structure of viruses, and mathematical modeling of infectious diseases helps provide insight into the disease dynamics and the design of intervention/vaccination programs. Scientists from different fields, extending from medicine and

molecular biology to computer science and applied mathematics, are teaming up for rapid assessment of potentially urgent situations [1–4].

This paper focusses on the computational aspects of simulating these mathematical models and parameter inference. Infectious diseases are often modeled using ordinary and partial differential equations (ODE and PDE). However, most models are non-linear in nature and cannot be solved analytically. Therefore a numerical method is generally used to provide approximate solutions. Performing parameter inference on these models typically requires a large number of evaluations with different parameter values, which can be an incredibly computationally expensive task.

Since their inception, graphics processing units (GPUs) have been transformed from common peripherals

\*Correspondence: [thomas.kovac@uhasselt.be](mailto:thomas.kovac@uhasselt.be)

†Equal contributors

<sup>1</sup>Center for Statistics, I-BioStat, Hasselt University, Agoralaan building D, 3590 Diepenbeek, Belgium

<sup>2</sup>Expertise Centre for Digital Media, Hasselt University, Wetenschapspark 2, 3590 Diepenbeek, Belgium

Full list of author information is available at the end of the article

into powerful devices that can be used for general-purpose programming [5–7]. GPUs are single-instruction multiple-data (SIMD) [8] devices. They contain chips that, in turn, contain hundreds of cores that allow hundreds of threads to be executed in parallel. CUDA [9], from NVIDIA, is a computing platform that exposes parallel compute power of NVIDIA GPUs to developers without a graphics background. However, it remains difficult to program GPUs for general-purpose use as the single-instruction, multiple-threads (SIMT) nature of GPUs does not allow all algorithms to be mapped onto a GPU.

Whereas GPUs are often exceptionally suited for solving big ODE/PDE problems (such as Computational Fluid Dynamics [10–12]), the models in infectious diseases are more troublesome; their smaller size makes it harder to optimally utilize all GPU cores and the SIMT nature introduces additional overhead which lowers performance. In contrast, simulating multiple small models simultaneously can be done efficiently on GPUs.

Inference of model parameters is usually done by maximizing the (log)likelihood such that the model accurately describes the data at hand. A number of local and global optimization methods can be used to this end. This paper focuses on particle swarm optimization (PSO): a method introduced by Kennedy et al. [13] for optimizing continuous non-linear functions. Based on the principles of bird flocking, fish schooling, and swarm theory, PSO can be implemented in a few lines of code.

Since the exchange of information between CPU and GPU is expensive [14], the common approach is to run all steps of the optimization algorithm on the GPU [15–18]. Instead, this paper proposes a heterogeneous approach for the following reasons:

- The CPU is a valuable resource that would otherwise be left idle.
- Implementing algorithms on the GPU requires thorough knowledge of hardware specifics for optimal performance.
- The sequential and branching nature of the algorithms makes them ill-suited for GPUs, since their SIMT architecture requires every thread to execute the same instruction at every moment.
- CPU implementation allows the use of scientific languages such as Python, Julia or MATLAB, significantly shortening development time.
- Several high-end implementations can be reused.

The contribution of this paper is twofold. First, an efficient GPU implementation for evaluating small-scale ODE models based on the Runge-Kutta method is presented and its scalability in number of threads and equations is investigated. Second, this paper proposes the

use of an asynchronous PSO implementation, for inferring parameters of an infectious disease model, that enables efficient utilization of both CPU and GPU resources.

### Related work

GPUs have evolved to such a degree that they have outpaced CPUs in terms of processing speed. As a result, they are being used for non-graphical applications such as high-order numerical integrations [19]. Amidst a plethora of numerical integration algorithms, Runge-Kutta methods are a family of implicit and explicit iterative methods, with a wide variety of orders and schemes [20]. Seen et al. [21] implemented a Runge-Kutta-Fehlberg (commonly denoted *RK45*) with adaptive step size on an NVIDIA GPU. The authors demonstrated that the GPU outperforms a CPU implementation, given that the problem dimensions are large enough, as in 200 equations, or more.

Niemeyer et al. [22, 23] developed GPU versions of the adaptive fifth-order Runge-Kutta-Cash-Karp (RKCK) method and stabilized second-order Runge-Kutta-Chebyshev (RKC) method, which are used for problems of non-stiff and greater stiffness nature, respectively. The authors came to the same conclusion; relative simple systems of ODEs, as in 512 equations, or less, limit the number of calculations performed on the GPU, resulting in the fact that the transfer of data to and from the GPU is expensive.

Murray [24] explains that a Runge-Kutta implementation with adaptive time steps is an example of a class of problems where the task-length for individual threads is variable. In this particular case, it is possible that the step size modification and error control can differ for each thread, which results in warp divergence and therefore loss in performance. To mitigate this complication, the author suggests that multiple data items could be bundled into each thread. When a task for one item is completed, a thread may advance immediately onto the next task.

In the last decade, several studies have been conducted on PSO methods in terms of performance and efficiency, new applications, and new variants of the algorithm. Both Koh et al. [25] and Venter et al. [26] concurrently, but independently, published work on a parallel and asynchronous implementation of the PSO algorithm. Both articles propose the use of a master-slave approach, where slave processors evaluate particles and the master conducts particle updates in terms of velocity and position.

Although this paper focuses on an asynchronous CPU implementation of PSO that ships off particles to the GPU for evaluation, the work of the following authors is included for completeness, as they concentrated on implementing a GPU version of PSO. Veronese et al. [15] were the first to experiment with the PSO algorithm and CUDA. With the entire, classical, PSO algorithm

ported to GPU using CUDA, they observed a significant reduction of the computing time compared to both C and MATLAB implementations. Wachowiak et al. [16] proposed an asynchronous GPU-based approach of PSO. Each particle is handled by a separate GPU thread, where they run a specified number of iterations after which they are resynchronized.

Mussi et al. [17] also present an asynchronous GPU implementation of the PSO algorithm. The authors aspire to overcome the drawbacks of asynchronous PSO imposed by the master-slave approach. In their method, the neighborhood is updated immediately after a particle is evaluated. The same authors compared their asynchronous implementation against their synchronous GPU version of PSO where they noticed speedups ranging from 5 to 35 times, depending on the problem's dimensions [27].

Hung et al. [18] propose a synchronous GPU implementation of the PSO method, meaning that the entire process is executed on GPU. Updating velocity, position, evaluating the given function, and updating best local and global candidate solutions are all implemented as separate CUDA kernels. This implies that, although the implementation is sped up by use of a GPU, the method is still synchronous.

Wende et al. [28, 29] observed that the GPU can handle large amounts of work, but that small-scale workloads are expensive to evaluate on a GPU. When doing so, the cost of shipping work towards and from the GPU is rather high. Using the Hyper-Q feature of NVIDIA's GPUs, the authors provided a single, shared, GPU of work using multiple CPU threads. This feat can assure that the resources of both the CPU and the GPU are efficiently used.

### Solving epidemiological models on GPUs

This paper proposes that infectious disease dynamic models are evaluated in parallel on the GPU. Since the interest lies with inferring parameters of infectious disease models, the following section outlines the characteristics of said models. Second, a comparison is made between the proposed approach and the previously mentioned ones, showing where performance is gained. Third, both the integration method and the right-hand of the ODEs can benefit from parallelization as vector operations and matrix-vector multiplications (i.e. calculating the force of infection) can easily be performed on GPU.

#### The SIR model

A basic model that is often used to study infectious disease dynamics in a population, is the Susceptible-Infected-Recovered or SIR model which describes the flow of individuals through these mutually exclusive disease states. In terms of the SIR model, the following set of partial differential equations (PDEs) can be used to model these dynamics,

$$\begin{cases} \frac{\partial S(a,t)}{\partial a} + \frac{\partial S(a,t)}{\partial t} = -(\lambda(a,t) + \mu(a))S(a,t), \\ \frac{\partial I(a,t)}{\partial a} + \frac{\partial I(a,t)}{\partial t} = \lambda(a,t)S(a,t) - (v + \mu(a))I(a,t), \\ \frac{\partial R(a,t)}{\partial a} + \frac{\partial R(a,t)}{\partial t} = vI(a,t) - \mu(a)R(a,t). \end{cases} \quad (1)$$

where  $S(a,t)$ ,  $I(a,t)$ , and  $R(a,t)$  are the age- and time-specific ( $a$  and  $t$ ) number of susceptibles, infected and recovered, respectively, with  $S(0,t) = B(t)$  the number of newborns at time  $t$ . In this paper, the basic SIR model is used, whereas in practice this model is commonly extended (e.g. Goeyvaerts et al. [30] use a SEIRS model). Such extensions typically exhibit the same dynamics and characteristics as the basic SIR model.

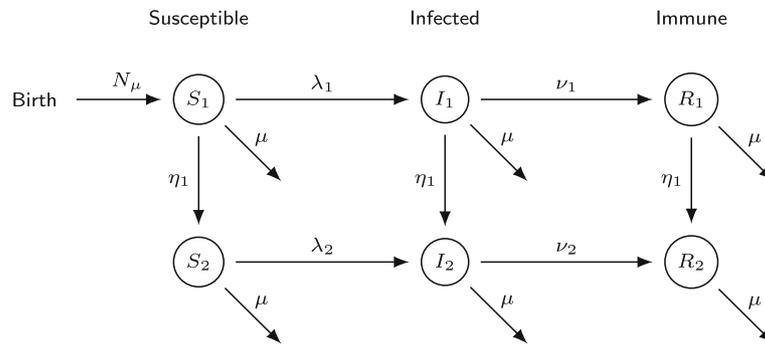
The force of infection  $\lambda(a,t)$  is given by the mass action principle:

$$\lambda(a,t) = \int_0^{+\infty} \beta(a,a',t)I(a',t)da'.$$

where  $\beta(a,a',t) \equiv \beta(a',a,t)$  is often assumed to be time-independent and governed by so-called Who Acquires Infection From Whom (WAIFW) matrices, i.e. mathematically convenient structures or by the social contact hypothesis (see e.g. [31]).

Solving a set of PDEs is not straightforward and depends on specific assumptions made for the different model parameters. In practice most PDE models are approximated using different methods in order to reduce the PDE model to a more solvable and workable model. Two well-known procedures are often used in mathematical epidemiology [32]: (1) The Cohort Age Structured (CAS) model replaces the set of PDEs with a set of ordinary differential equations (ODEs) by considering  $K$  compartmental models representing  $K$  age-groups and using continuous transitions from one age-group to the next (see Fig. 1;  $\eta_1$ ). (2) In a Realistic Age Structured (RAS) model individuals change status from S to I and from I to R during one year (assuming age-groups of one year) after which they instantaneously move to the next age-group (see e.g. [30]). It is often said that a RAS-model better reflects infectious disease dynamics because children switch grades in school generally at the same moment during the year, and only once per year (see e.g. [33]). For a more in depth discussion about the advantages and disadvantages of the different methods to discretize age-structured PDE models, the reader is referred to [32, 34, 35].

This paper focusses mostly on the CAS model, but the RAS model would be fairly similar. The CAS model uses a backward in space finite difference scheme [36], comparable to the method of characteristics [37], essentially breaking down the PDE into the following set of coupled ODEs:



**Fig. 1** Flow diagram for the CAS implementation of the SIR model. The individuals enter the susceptible class, then move to the infected class (at rate  $\lambda$ ) and after recovering they move into the immune class (at rate  $\nu$ ).  $\eta_i$  is the transition rate from category  $a_i$  to  $a_{i+1}$ , and  $\mu$  is the mortality rate in this diagram

$$\begin{cases} \frac{S(a,t) - S(a-\Delta a,t)}{\Delta a} + \frac{\partial S(a,t)}{\partial t} = -(\lambda(a,t) + \mu(a))S(a,t), \\ \frac{I(a,t) - I(a-\Delta a,t)}{\Delta a} + \frac{\partial I(a,t)}{\partial t} = \lambda(a,t)S(a,t) - (\nu + \mu(a))I(a,t), \\ \frac{R(a,t) - R(a-\Delta a,t)}{\Delta a} + \frac{\partial R(a,t)}{\partial t} = \nu I(a,t) - \mu(a)R(a,t). \end{cases} \quad (2)$$

Note that using this discretization, the force of infection can be rewritten as:

$$\lambda(a,t) = \sum_{a'} \beta(a,a')I(a',t). \quad (3)$$

To ensure stability of the scheme, a necessary and sufficient condition is the Courant-Friedrichs-Lewy condition [36] which requires  $\Delta t \leq \Delta a$ .

### Runge-Kutta-Fehlberg on GPU

With many methods to choose from, the Runge-Kutta-Fehlberg method, commonly known as *RK45*, is characterized as being an integration method that provides the most bang for the buck [20]. In his paper [38], Fehlberg describes a fifth-order method with six function evaluations where another combination of the same six functions gives a fourth-order method. The difference between these estimates approximate the truncation error, which in turn adjusts the step size. Related work shows that there have been successful attempts at porting a Runge-Kutta method to GPU. However, in most of those implementations, each GPU thread evaluates a system of differential equations of varying task-length, due to analyzing different parameter candidates for inference and given the SIMT nature of GPU, it is difficult to minimize performance loss. The following sections will outline our approach of implementing an efficient *RK45* implementation on GPU.

### Using a block of threads

The proposed method differs from existing techniques in that a block of threads, mapped on a single streaming multiprocessor (SM), jointly integrate a single ODE while

different blocks are working on other ODEs. This has the advantage that threads can communicate via shared memory instead of global memory (approximately 60x faster) [39]. However this technique only works for small systems that fit in shared memory. The reduction step in the *RK45* method would normally require a synchronization barrier, but can be avoided when using a block of threads through the use of warp shuffle functionality of CUDA [39]. As a result the threads can execute the *RK45* method in lock-step.

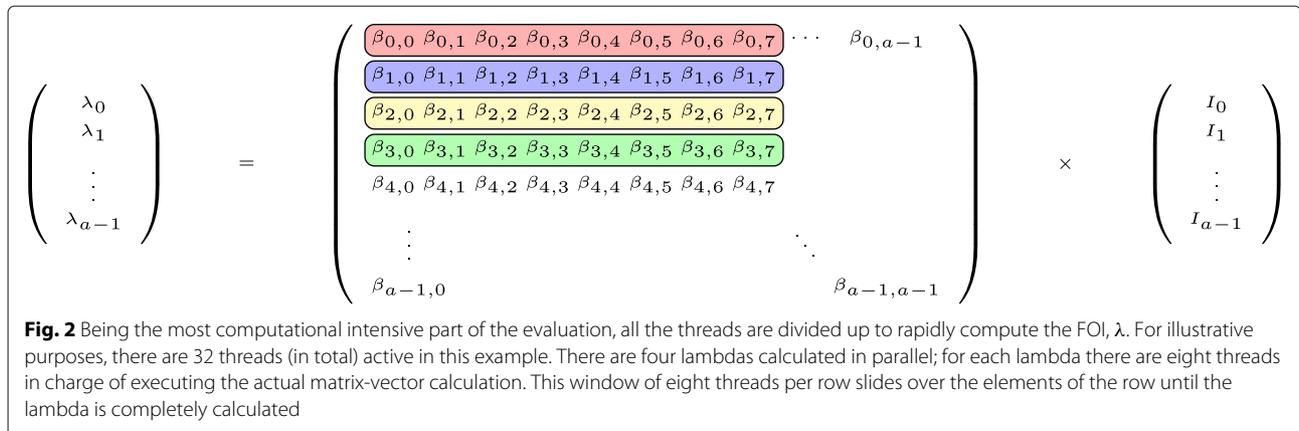
### Calculating the force of infection

Another integral part of the integration process is the right-hand side of the ODE, which can also benefit from parallelization. Multiplying the social contact matrix  $\beta$  with a vector of infected  $I$ , results in the force of infection (FOI)  $\lambda$ , as depicted in Fig. 2 and Eq. 3. Given that at each iteration of the integration process  $I$  is updated based on an update of  $\lambda$ , and this multiplication must be performed several times, it is the most computational intensive part of the integration process.

Figure 2 shows that available threads are divided up in order to calculate the FOI. The division scheme, describing how many threads are used per row, is chosen based on the GPU architecture. When calculating a single  $\lambda$  value, the warp shuffle feature of CUDA is used to perform reductions.

### Asynchronous parameter inference

The next, and final, step is to infer model parameters so that the SIR model accurately describes the observed data by maximizing the (log)likelihood of the model, also called maximum likelihood estimation (MLE). This entails that the model is integrated using candidate parameters, applying the ODE integration technique introduced in previous sections. This paper proposes the use of an asynchronous particle swarm method for parameter inference with the goal of fully utilizing both CPU and GPU.



**Particle swarm optimization**

Particle swarm optimization, introduced by Kennedy et al. in 1995 [13], is based on the principles of bird-flocking, fish schooling, and swarm theory. Individual members of the society can profit from the discoveries made by other members during the search for food. The same principle is applied to explore the parameter space of (non-linear) problems.

Particles have both a position and velocity which are updated at every iteration using the particles' best known local ( $P_{best}$ ) and global position ( $g_{best}$ ). The vanilla method decomposes into two steps, as described in Algorithm 1, namely to first initialize all particles and then update the particles' position and velocity until a termination criterion has been met.

Each particle represents candidate parameters for the SIR model and will be evaluated by first integrating the ODE using the proposed GPU implementation of the RK45 method and then passing the result to the (log)likelihood function. While the evaluation can be performed in parallel, alternating between GPU and CPU results in inefficient resource utilization. Only one processor is making progress at any given time. In addition, if some particles take longer to evaluate, resource utilization drops further.

**Asynchronous particle swarm optimization**

The vanilla PSO algorithm (Algorithm 1) requires that all particles have updated their velocity and position before advancing to the next iteration. On the other hand, asynchronous PSO [25, 26, 40] removes this barrier with the goal of keeping all processing units from idling. The key to implement an asynchronous PSO algorithm is to separate the update actions for individual particles from those associated with the swarm as a whole. While the asynchronous algorithm might require additional iterations due to some particles running ahead of others, the convergence rate is generally reported as comparable and the algorithm significantly outperforms its synchronous

**Algorithm 1 Particle Swarm Optimization**

```

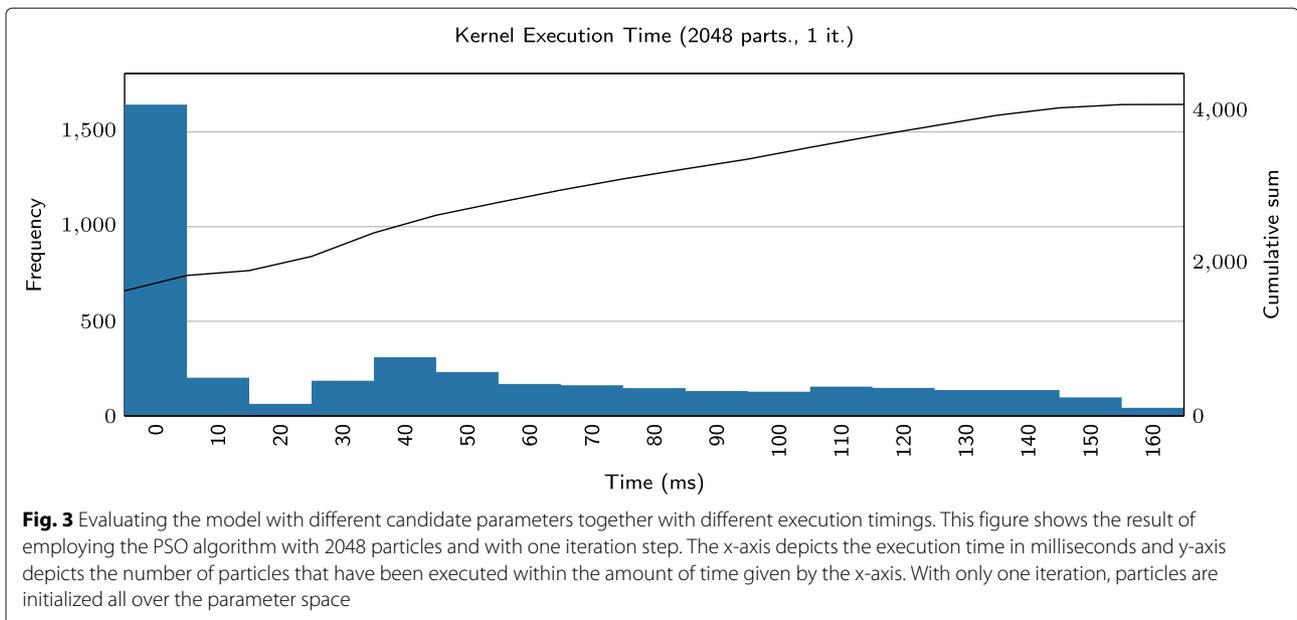
1: procedure INITPARTICLES
2:   for every particle P do
3:      $P_{pos} \leftarrow RandomPosition()$ 
4:      $P_{vel} \leftarrow RandomVelocity()$ 
5:      $P_{best} \leftarrow P_{position}$ 
6:     if  $cost(P_{best}) < cost(g_{best})$  then
7:        $g_{best} \leftarrow P_{best}$ 
8: procedure UPDATEPARTICLES
9:   while termination criterion is not met do
10:    for every particle P do
11:       $P_{vel} \leftarrow UpdateVelocity(P_{vel}, P_{best}, g_{best})$ 
12:       $P_{pos} \leftarrow UpdatePosition(P_{pos}, P_{vel})$ 
13:    for every particle P do
14:      if  $cost(P_{pos}) < cost(P_{best})$  then
15:         $P_{best} \leftarrow P_{pos}$ 
16:        if  $cost(P_{best}) < cost(g_{best})$  then
17:           $g_{best} \leftarrow P_{best}$ 

```

counterpart in terms of parallel efficiency [25, 26]. This is especially true when the evaluation time of a particle depends on its position, which is the case for numerical integration of ODEs (as also noted by Murray [24]). Figure 3 shows a histogram of the execution times for the evaluations of 2048 particles in the first iteration of the PSO algorithm. As the execution time varies between 0ms and 160ms, the synchronous algorithm will have significant idling of processing units. In the asynchronous algorithm, faster particles can advance to the next iteration and will positively influence the slower ones, as information is shared between the whole population of particles.

**Heterogeneous approach**

The key idea to fully utilize both CPU and GPU resources is to produce many concurrent GPU workloads which



as a whole can fill up the GPU to capacity and asynchronously offload them such that the CPU can concurrently pre/post-process the workloads. The Concurrent Kernel Execution (CKE) and Hyper-Q features introduced by NVIDIA with the Fermi and Kepler architectures, respectively, enables this by allowing multiple CPU cores to simultaneously launch concurrent workloads on a single, shared, GPU. The Hyper-Q feature allows 32 simultaneous, hardware-managed connections (or work queues), compared to the single work queue available on the Fermi architecture.

When dealing with imbalanced workloads, it is advantageous to have multiple kernels executing concurrently instead of a single monolithic kernel as the latter would idle GPU resources while finishing the longest of tasks. CKE on the other hand allows the GPU to continue execution with the next workloads. Since offloading to GPU can be costly due to data transfer, kernel launching and synchronization [14], particles are grouped into batches to amortize these costs. Given that execution time of a batch is determined by the slowest particle in the batch, there is a trade-off between batching and the ability to deal with the imbalance.

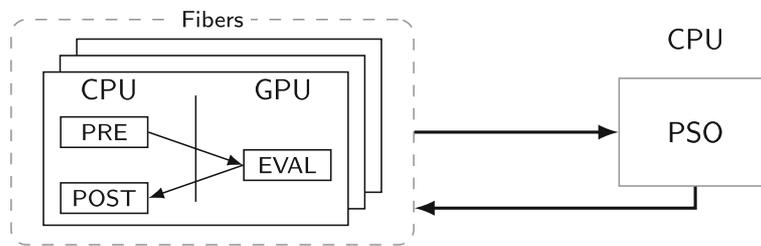
It is important to note that, while Hyper-Q allows offloading of multiple kernels, the newer generation of GPUs still only have two copy engines. This implies that only two data transfer operations can happen simultaneously: one transfer from device to host memory and one in the opposite direction. This could result in a transfer bottleneck with a drop in performance as a result.

The CPU implementation of the particle swarm algorithm has two goals: maximize CPU utilization and separate the optimization algorithm from GPU details. Ideally, the optimization algorithm should be completely oblivious to anything GPU related such that it is easy to reuse existing implementations and switch between different algorithms.

The former is accomplished by multiple CPU threads processing particles in parallel thus utilizing all cores on the host and through the use of fibers (or coroutines [41]). Fibers are light-weight threads of execution that enable cooperative multitasking. This means that a fiber must yield its execution to enable the execution of another. In the proposed method, each CPU thread handles one or more fibers (see Fig. 4). Whenever the thread must wait for the GPU to finish, it switches to another fiber and continues processing different particles: either pre-processing, offloading to GPU or computing the likelihood. The use of fibers also enables a single CPU core to process multiple particles at the same time and to occupy the 32 Hyper-Q connections.

While the use of fibers is not necessarily required to maximize CPU usage, it makes the implementation of the PSO algorithm much easier and free of any GPU related details. The alternative would be to manage the Hyper-Q work queues manually inside the algorithm. The proposed implementation consolidates such details in the GPU implementation.

For every Hyper-Q work queue, the implementation also keeps several work packages containing CPU pinned memory and GPU device memory such that it can avoid expensive memory allocation for every offload. These



**Fig. 4** The PSO algorithm is run on CPU side, while the evaluation of the particles is handled by the GPU. CPU cores send batches of particles to the GPU asynchronously in order to increase performance. Before a batch is sent to the GPU, some pre-processing can be performed, as well as eventual post-processing when the evaluation is finished

packages are placed in a concurrent first-in first-out data structure from which the CPU threads can acquire them whenever an evaluation is offloaded. In this case, the `concurrent_queue` implementation of the TBB template library [42], developed by Intel, is used.

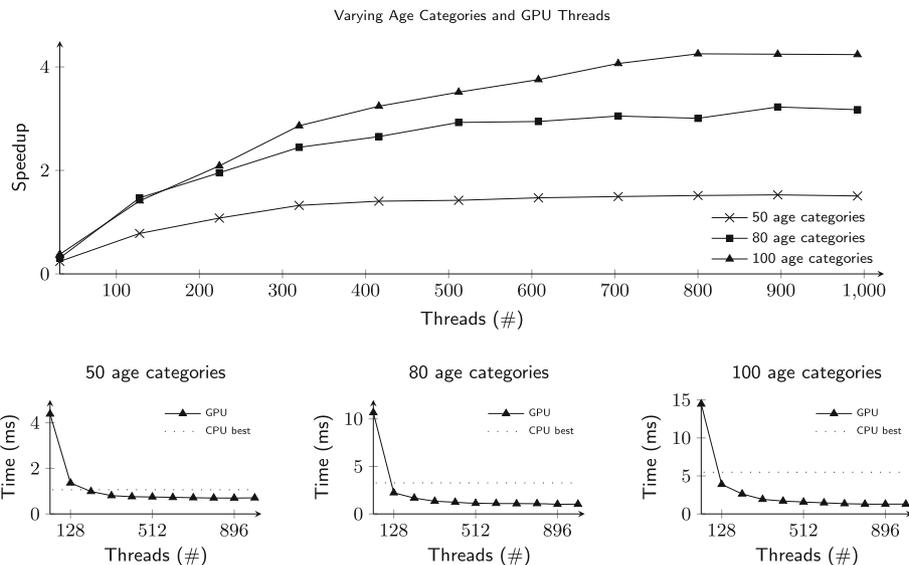
**Results and discussion**

In this section, the performance and behavior of the *RK45* implementation as well as the asynchronous parameter inference using a heterogeneous architecture is described. The experiments were conducted on a machine with four eight-core Intel Westmere processors running at 2.67GHz and an NVIDIA GeForce GTX TITAN X. The machine runs the GNU/Linux operating system and CUDA 8.0 was used for programming the GPU.

**Scalability and performance of *RK45* on GPU**

Figure 5 shows the speedup in terms of number of GPU threads used of the *RK45* implementation when integrating a single ODE as well as a performance improvement of between 1.5 and 4.23 times compared to a CPU implementation. The number of operations of this model is in the order of  $\mathcal{O}(n^2)$ , meaning that when comparing 50 and 100 age categories, the latter has 4 times as many operations. As noted by previous work [21–23], small-scale ODE integration cannot fully utilize the GPU. By increasing the amount of age categories, the number of ODE equations grows proportionally and allows more GPU threads to be effectively used.

Filling the GPU up to capacity, however, requires an even larger ODE system. In contrast, the approach taken by this paper is to integrate many small-scale ODEs at the



**Fig. 5** The bottom row depicts the model execution timings on GPU when using three different amounts of age categories. The x-axis depicts the amount of GPU threads used when evaluating an ODE, while the y-axis depicts the execution timing in milliseconds. The number of operations is in the order of  $\mathcal{O}(n^2)$ , meaning that when comparing 50 and 100 age categories, the latter has 4 times as many operations. One can see that when there are more age categories available, the better the GPU will perform when compared to the CPU. The top row depicts the speedup of the GPU when compared with the best CPU timing for each of the three age category settings

same time. Figure 6 shows that the execution time grows linearly for both CPU and GPU as the number of ODEs increases. Thus the GPU implementation remains up to 4 times faster, regardless of the amount of work.

The following experiments always use the introduced SIR model with 100 age-categories as it is commonly used and represents a realistic range of ages.

### Particle swarm optimization

To demonstrate the performance of the heterogeneous approach compared to the CPU-only implementation, both synchronous and asynchronous versions of the PSO algorithm are run with 256 and 2048 particles. Execution is terminated after 100 iterations, under the assumption that convergence rates are the same for both versions, in order to effectively compare execution times. All tunable parameters, such as number of CPU/GPU threads and fibers, are explored and the best configuration is reported.

Results are illustrated in Fig. 7. The synchronous version benefits from the acceleration of the ODE integration on GPU. However, as shown in Fig. 3, integration times can vary significantly for different candidate values. The trade-off in terms of batch size is clearly visible, even in the synchronous case. Decreasing the batch size allows CPU threads to commence post-processing more quickly, instead of waiting for the slowest integration to finish. As the barrier prohibits fast batches of particles to continue to the next iteration, this benefit is not substantial. The maximal speedup measured was 6.2 and 5.3 when using 256 and 2048 particles, respectively.

The asynchronous implementation removes the latter problem, permitting some particles to run ahead of slow particles, resulting in a significant speedup and the batch size playing a more important role. The trade-off between the ability to deal with imbalance and the overhead of offloading multiple batches is clearly visible; a batch size that is too small introduces too much overhead. Maximal

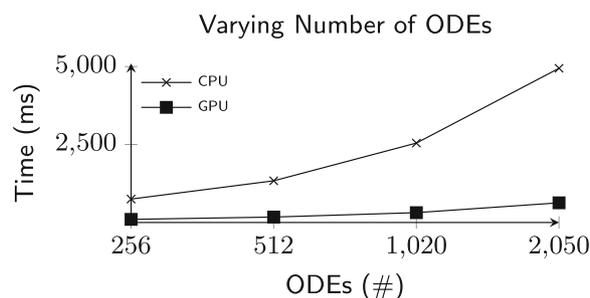
speedup measured for the asynchronous implementation was 12 and 10 when using 256 and 2048 particles, respectively.

Figure 8 shows that the use of fibers has no significant influence on execution time. For example, 32 CPU threads, each one fiber, performs equally well when compared to using one CPU thread with 32 fibers. The same can be said over a moderate, eight-core system with four fibers.

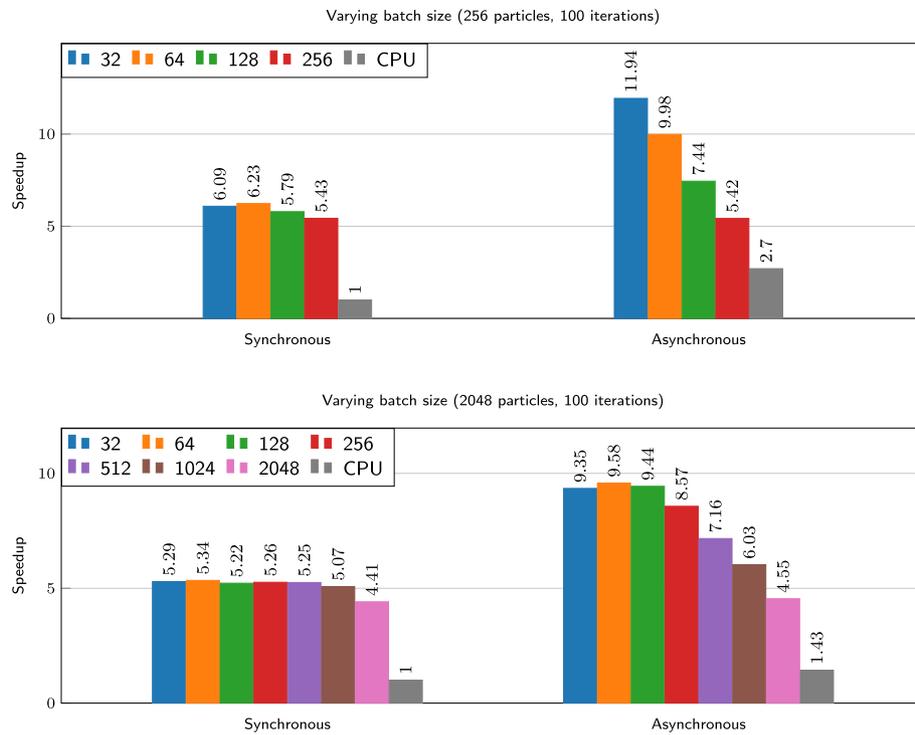
### Conclusion

A crucial part in performing parameter inference for epidemiological models is integrating a set of ODEs. The efficient implementation of *RK45* on GPU is capable of integrating up to 4 times faster than a CPU implementation. Concurrent offloading of many small-scale ODEs can fill up the GPU to capacity while maintaining this performance. The efficiency of the implemented *RK45* method can also be improved by making it customized to the SIR model instead of the generic approach proposed in this paper.

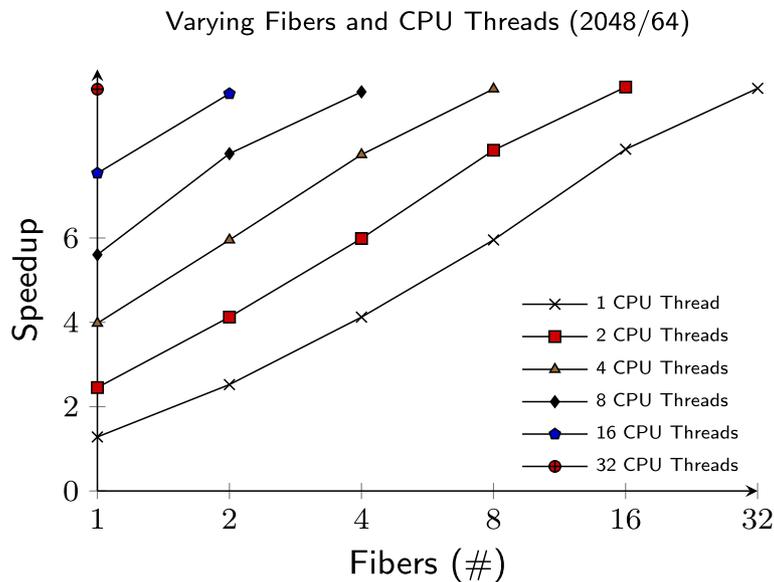
The proposed method keeps optimization methods on the CPU. Such a heterogeneous approach to asynchronous particle swarm optimization, as described in this paper, provides a way to perform parameter inference while efficiently utilizing resources of both CPU and GPU and decreases execution time. Evaluating all particles at once, the overall execution time is determined by the slowest evaluation. Grouping particles into batches mitigates this variance in execution time and increases performance. The removal of the synchronization barrier, prohibiting particles to continue to the next iteration, allows certain batches to run ahead of other batches. Note, however, that a trade-off has to be made between the ability to deal with imbalance and the overhead of offloading multiple batches of particles. The optimal batch size in our experiments achieves a speedup up to 12 times.



**Fig. 6** Each line represents the execution time for CPU and GPU when integrating an increasing number of ODEs. The x-axis depicts the number of ODEs used and the y-axis depicts the execution time in milliseconds. Both approaches scale linearly when given more work to process with the GPU remaining competitively faster



**Fig. 7** Employing the heterogeneous, asynchronous, PSO algorithm using 256 and 2048 particles, respectively. Each bar represents the execution time in milliseconds and each color represents a different batch size. The figure depicts that the asynchronous approach not only benefits from a fast integration method on GPU, but keeping both CPU and GPU busy with useful work is more preferable when compared to the synchronous approach. Given possible imbalance when evaluating ODEs, it is advisable to evaluate the particles in small batches



**Fig. 8** Given that batches of particles can be evaluated asynchronously, one can ship multiple batches of particles to the GPU in parallel. Once a CPU thread ships off a batch of particles to the GPU, it needs to wait for it to be finished before continuing to work. Using fibers, a CPU thread can process multiple batches at the same time and therefore increases performance. The x-axis depicts the amount of fibers per CPU thread and the y-axis depicts the speedup of the GPU compared to the best CPU timing. A total of 2048 particles were used with 64 particles per batch

The proposed contributions enable researchers, for example, to deal with urgent situations like epidemic outbreaks, where a rapid assessment of intervention strategies is required. Future research should ascertain how these contributions perform on newer GPU architectures as well as how they perform on real-life epidemiological scenarios. Also, the proposed approach has been programmed to work with all available GPUs on a host system. However, such a setup has not been evaluated by the authors and is well worth looking into.

#### Abbreviations

CAS: Cohort age structured; CPU: Central processing unit; CKE: Concurrent kernel execution; FOI: Force of infection; GPU: Graphics processing unit; ODE: Ordinary differential equation; PSO: Particle swarm optimization; PDE: Partial differential equation; RAS: Realistic age structured; RK45: Runge-Kutta-Fehlberg; SIR: Susceptible-infected-recovered; SIMD: Single-instruction multiple-data; SIMT: Single-instruction multiple-threads; WAIFW: Who acquires infection from whom; SM: Streaming multiprocessor; MLE: Maximum likelihood estimation

#### Acknowledgements

TK acknowledges support from a Methusalem research grant from the Flemish government. NH gratefully acknowledges support from the University of Antwerp scientific chair in Evidence-Based Vaccinology, financed by a gift from Pfizer (2009-2017) and GSK (2017). This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 682540 - TransMID) and from the Special Research Fund of Hasselt University.

#### Availability of data and materials

Belgian demographic data was used to determine the initial age-specific population distribution as well as an age-specific daily mortality rate. Both datasets were obtained from Eurostat (<http://ec.europa.eu/eurostat/data/database>), then click on "Tables by themes", "Population and social conditions", "Population", "Demography", "Population", and "Population on 1 January".

#### Authors' contributions

TK and TH both conceived, designed, and performed the experiments and analyzed the resulting data. NH contributed the materials and analysis tools. All authors wrote the paper. All authors read and approved the final manuscript.

#### Ethics approval and consent to participate

Not applicable.

#### Consent for publication

Not applicable.

#### Competing interests

The authors declare that they have no competing interests.

#### Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

#### Author details

<sup>1</sup>Center for Statistics, I-BioStat, Hasselt University, Agoralaan building D, 3590 Diepenbeek, Belgium. <sup>2</sup>Expertise Centre for Digital Media, Hasselt University, Wetenschapspark 2, 3590 Diepenbeek, Belgium. <sup>3</sup>Centre for Health Economic Research and Modelling Infectious Diseases, Vaccine and Infectious Disease Institute, University of Antwerp, Universiteitsplein 1, 2610 Wilrijk, Belgium.

Received: 28 July 2017 Accepted: 5 March 2018

Published online: 16 March 2018

#### References

- Colizza V, Barrat A, Barthelemy M, Valleron A-J, Vespignani A. Modeling the worldwide spread of pandemic influenza: baseline case and containment interventions. *PLoS Med.* 2007;4(1):13. <https://doi.org/10.1371/journal.pmed.0040013>.

- Stohr K, Esveld M. PUBLIC HEALTH: enhanced: will vaccines be available for the next influenza pandemic? *Science.* 2004;306(5705):2195–6. <https://doi.org/10.1126/science.1108165>.
- Gyawali N, Bradbury RS, Taylor-Robinson AW. The global spread of Zika virus: is public and media concern justified in regions currently unaffected? *Infect Diseases Poverty.* 2016;5:37. <https://doi.org/10.1186/s40249-016-0132-y>.
- Siettos C, Anastassopoulou C, Russo L, Grigoras C, Mylonakis E. Modeling the 2014 ebola virus epidemic - agent-based simulations, temporal analysis and future predictions for Liberia and Sierra Leone. *PLoS Currents.* 2015;7. <https://doi.org/10.1371/currents.outbreaks.8d5984114855fc425e699e1a18cddc6c9>.
- Owens JD, Luebke D, Govindaraju N, Harris M, Krüger J, Lefohn AE, Purcell TJ. A survey of general-purpose computation on graphics hardware. *Comput Graph Forum.* 2007;26(1):80–113. doi: [10.1111/j.1467-8659.2007.01012.x](https://doi.org/10.1111/j.1467-8659.2007.01012.x).
- Buck I, Govindaraju N, Harris M, Krüger J, Lefohn A, Luebke D, Purcell T, Wooley C. Gpgpu: General purpose computation on graphics hardware. In: ACM SIGGRAPH 2004 Course Notes. SIGGRAPH '04. New York: ACM; 2004. doi: [10.1145/1103900.1103933](https://doi.org/10.1145/1103900.1103933). <http://doi.acm.org/10.1145/1103900.1103933>.
- Lindholm E, Kilgard MJ, Moreton H. A User-programmable Vertex Engine. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '01. New York: ACM; 2001. p. 149–58. <https://doi.org/10.1145/383259.383274>. <http://doi.acm.org/10.1145/383259.383274>.
- Flynn MJ, Rudd KW. Parallel architectures. *ACM Computing Surveys (CSUR).* 1996;28(1):67–70.
- NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. Santa Clara, CA: NVIDIA Corporation; 2007.
- Elsen E, LeGresley P, Darve E. Large calculation of the flow over a hypersonic vehicle using a gpu. *J Comput Phys.* 2008;227(24):10148–61. <https://doi.org/10.1016/j.jcp.2008.08.023>.
- Niemeyer KE, Sung CJ. Accelerating Reactive-Flow Simulations Using Graphics Processing Units. Aerospace Sciences Meetings. Grapevine: American Institute of Aeronautics and Astronautics; 2013. doi: [10.2514/6.2013-371](https://doi.org/10.2514/6.2013-371). <https://doi.org/10.2514/6.2013-371>.
- Liu Y, Deng L. In: Wang G, Zomaya A, Martinez Perez G, Li K, editors. Acceleration of CFD Engineering Software on GPU and MIC. Cham: Springer; 2015. pp. 835–48.
- Kennedy J, Eberhart R. Particle Swarm Optimization. In: Proceedings of IEEE International Conference on Neural Networks, vol. 4. Perth: IEEE; 1995. p. 1942–48. doi: [10.1109/ICNN.1995.488968](https://doi.org/10.1109/ICNN.1995.488968).
- Lustig D, Martonosi M. Reducing gpu offload latency via fine-grained cpu-gpu synchronization. In: High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium On. Washington, DC: IEEE Computer Society; 2013. p. 354–65.
- Veronese L, Krohling RA. Swarm's Flight: Accelerating the Particles Using C-CUDA. In: Proceedings of the Eleventh Conference on Congress on Evolutionary Computation. Piscataway: IEEE Press; 2009. p. 3264–70. <http://dl.acm.org/citation.cfm?id=1689599.1690035>.
- Wachowiak MP, Foster AEL. GPU-based asynchronous global optimization with particle swarm. *J Phys Conf Series.* 2012;385(1):12012.
- Mussi L, Nashed YSG, Cagnoni S. GPU-based Asynchronous Particle Swarm Optimization. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. GECCO '11. New York: ACM; 2011. p. 1555–62. <https://doi.org/10.1145/2001576.2001786>. <http://doi.acm.org/10.1145/2001576.2001786>.
- Hung Y, Wang W. Accelerating parallel particle swarm optimization via GPU. *Optim Methods Softw.* 2012;27(1):33–51. <https://doi.org/10.1080/10556788.2010.509435>.
- Hawick K, Playne DP, Johnson M. Numerical Precision and Benchmarking Very-High-Order Integration of Particle Dynamics on GPU accelerators. In: Proceedings of the 2011 International Conference on Computer Design (CDES'11). 2011. p. 83–89.
- Press WH, Teukolsky SA, Vetterling WT, Flannery BP. Numerical Recipes 3rd Edition: The Art of Scientific Computing, 3rd edn. New York: Cambridge University Press; 2007.
- Seen WM, Gobithaasan RU, Miura KT, Ismail MT, Ahmad S, Rahman RA. GPU Acceleration of Runge Kutta-Fehlberg and Its Comparison with Dormand-Prince Method. *AIP Conf Proc.* 2014;1605(1):16–21. <https://doi.org/10.1063/1.4887558>.

22. Niemeyer KE, Sung CJ. Accelerating moderately stiff chemical kinetics in reactive-flow simulations using GPUs. *J Comput Phys*. 2014;256:854–71. <https://doi.org/10.1016/j.jcp.2013.09.025>.
23. Niemeyer KE, Sung CJ. In: Kindratenko V, editor. GPU-Based Parallel Integration of Large Numbers of Independent ODE Systems. Cham: Springer; 2014, pp. 159–82. [http://dx.doi.org/10.1007/978-3-319-06548-9\\_8](http://dx.doi.org/10.1007/978-3-319-06548-9_8).
24. Murray L. GPU Acceleration of Runge-Kutta Integrators. *IEEE Trans Parallel Distributed Syst*. 2012;23(1):94–101. <https://doi.org/10.1109/TPDS.2011.61>.
25. Koh BI, George AD, Haftka RT, Fregly BJ. Parallel Asynchronous Particle Swarm Optimization. *Int J Numeric Methods Eng*. 2006;67(4):578–95. <https://doi.org/10.1002/nme.1646>.
26. Venter G, Sobieszcanski-Sobieski J. Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. *J Aerospace Comput Inf Commun*. 2006;3(3):123–37.
27. Mussi L, Daolio F, Cagnoni S. Evaluation of parallel particle swarm optimization algorithms within the CUDA architecture. *Inf Sci*. 2011;181(20):4642–57. <https://doi.org/10.1016/j.ins.2010.08.045>.
28. Wende F, Cordes F, Steinke T. On Improving the Performance of Multi-threaded CUDA Applications with Concurrent Kernel Execution by Kernel Reordering. In: 2012 Symposium on Application Accelerators in High Performance Computing. Chicago: IEEE; 2012. p. 74–83.
29. Wende F, Steinke T, Cordes F. Multi-threaded Kernel Offloading to GPGPU Using Hyper-Q on Kepler Architecture. Technical Report 14-19, ZIB, Takustr.7, 14195 Berlin; 2014.
30. Goeyvaerts N, Willem L, Kerckhove KV, Vandendijck Y, Hanquet G, Beutels P, Hens N. Estimating dynamic transmission model parameters for seasonal influenza by fitting to age and season-specific influenza-like illness incidence. *Epidemics*. 2015;13:1–9. <https://doi.org/10.1016/j.epidem.2015.04.002>.
31. Hens N, Shkedy Z, Aerts M, Faes C, Van Damme P, Beutels P. Modeling Infectious Disease Parameters Based on Serological and Social Contact Data, 1st edn. *Statistics for Biology and Health*, vol. 63. Springer Heidelberg Dordrecht London New York: Springer New York; 2012. p. 300. <https://doi.org/10.1007/978-1-4614-4072-7>.
32. Vynnycky E, White RG. *An Introduction to Infectious Disease Modelling*. Oxford: Oxford University Press; 2010.
33. Schenzle D. An age-structured model of pre- and post-vaccination measles transmission. *IMA J Math Appl Med Biol*. 1984;1(2):169–91.
34. Capasso V. *Mathematical Structures of Epidemic Systems*. Springer-Verlag Berlin Heidelberg: Springer; 2008.
35. Keeling MJ, Rohani P. *Modeling infectious diseases in humans and animals*. Princeton: Princeton University Press; 2008.
36. Strikwerda JC. *Finite Difference Schemes and Partial Differential Equations*. Belmont: Wadsworth Publ. Co.; 1989.
37. Courant R, Isaacson E, Rees M. On the solution of nonlinear hyperbolic differential equations by finite differences. *Commun Pure Appl Math*. 1952;5(3):243–55. <https://doi.org/10.1002/cpa.3160050303>.
38. Fehlberg E. Low-Order Classical Runge-Kutta Formulas with Step Size Control and their Application to Some Heat Transfer Problems. Technical report, NASA Marshall Space Flight Center, Huntsville, AL, United States 1969.
39. CUDA C Programming Guide. <https://docs.nvidia.com/cuda/cuda-c-programming-guide/>. Accessed 20 Jan 2017.
40. Schutte JF, Reinbolt JA, Fregly BJ, Haftka RT, George AD. Parallel Global optimization with the particle swarm algorithm. *Int J Numeric Methods Eng*. 2004;61(13):2296–315. <https://doi.org/10.1002/nme.1149>.
41. Conway ME. Design of a separable transition-diagram compiler. *Commun ACM*. 1963;6(7):396–408. <https://doi.org/10.1145/366663.366704>.
42. Threading Building Blocks. <https://www.threadingbuildingblocks.org/>. Accessed 20 Jan 2017.

Submit your next manuscript to BioMed Central and we will help you at every step:

- We accept pre-submission inquiries
- Our selector tool helps you to find the most relevant journal
- We provide round the clock customer support
- Convenient online submission
- Thorough peer review
- Inclusion in PubMed and all major indexing services
- Maximum visibility for your research

Submit your manuscript at  
[www.biomedcentral.com/submit](http://www.biomedcentral.com/submit)

