Geosci. Model Dev. Discuss., 5, 2179–2214, 2012
www.geosci-model-dev-discuss.net/5/2179/2012/
doi:10.5194/gmdd-5-2179-2012

Geoscientific Model Development Discussions

Porting marine ecosystem model spin-up using transport matrices to GPUs

E. Siewertsen[1], **J. Piwonski**[2], **and T. Slawig**[2]

[1]Institute for Computer Science, Christian-Albrechts Universität zu Kiel, 24098 Kiel, Germany
[2]Institute for Computer Science and Kiel Marine Science – Centre for Interdisciplinary Marine Science, Cluster The Future Ocean, Christian-Albrechts Universität zu Kiel, 24098 Kiel, Germany

Received: 19 July 2012 – Accepted: 20 July 2012 – Published: 31 July 2012

Correspondence to: T. Slawig (ts@informatik.uni-kiel.de)

Published by Copernicus Publications on behalf of the European Geosciences Union.

Abstract

We have ported an implementation of the spin-up for marine ecosystem models based on the "Transport Matrix Method" to graphics processing units (GPUs). The original implementation was designed for distributed-memory architectures and uses the PETSc library that is based on the "Message Passing Interface (MPI)" standard. The spin-up computes a steady seasonal cycle of the ecosystem tracers with climatological ocean circulation data as forcing. Since the transport is linear with respect to the tracers, the resulting operator is represented in so-called "transport matrices". Each iteration of the spin-up involves two matrix-vector multiplications and the evaluation of the used biogeochemical model. The original code was written in C and Fortran. On the GPU, we use the CUDA standard, a specialized version of the PETSc toolkit and a CUDA Fortran compiler. We describe the extensions to PETSc and the modifications of the original C and Fortran codes that had to be done. Here we make use of freely available libraries for the GPU. We analyze the computational effort of the main parts of the spin-up for two exemplary ecosystem models and compare the overall computational time to those necessary on different CPUs. The results show that a consumer GPU can beat a significant number of cluster CPUs without further code optimization.

1 Introduction

This work is motivated by the usually huge effort that is needed when computing steady annual cycles (or, mathematically speaking, periodic solutions) of spatially three-dimensional marine ecosystem models. In most cases this is done by "spinning up" the model, i.e. by performing a time-stepping algorithm with climatological, periodic forcing data until the steady cycle is reached, at least up to a certain tolerance. This can take a huge number of iterations, in typical cases about 3000 to 5000 yr, each of which involves thousands of time steps (e.g. 2880 steps for a three-hour step-size). Thus the overall number of iterations may may be in the range of 10^6 to 10^7. When

GMDD
5, 2179–2214, 2012

aiming at parameter optimization or sensitivity studies, the spin-up process has to be repeated several times, and thus in these cases a reduction of the computational time of a single spin-up run is even more important.

There are several strategies to reduce this computational effort. The following ones are more or less independent from each other: one of them is of course parallelization, usually by domain decomposition methods. The second one is to use precomputed "transport matrices" (see Khatiwala, 2007) that represent the (eventually linearized) tracer transport scheme applied in an ocean model. Usually monthly averaged matrices for the explicit and the implicit parts of the ocean tracer transport operator are used. In the ecosystem spin-up, these averaged matrices are then interpolated accordingly in every time step. With this method, the transport part of the ecosystem model reduces to matrix-vector multiplications, whereas the biogeochemical source-minus-sink terms are evaluated separately. A third way to reduce computational effort is to replace the standard spin-up (which, in mathematically terms, is a fixed-point iteration) by variants of Newton's method, which have higher convergence rates.

In this work we start from an implementation of a spin-up that applies the first two strategies: it uses transport matrices generated by the MITgcm ocean model, see M.I.T. (2012). The tracer transport operator is thus pre-computed by twelve pairs of implicit and explicit matrices, each pair representing an average "climatological" month. The matrices are stored in a sparse format, and are appropriate to perform matrix-vector multiplications on distributed memory hardware using MPI. For this purpose, we use the PETSc library. The main advantages of this toolkit is that all MPI calls are hidden in built-in functions, and that optimized functions for matrix-vector operations (and more) already exist. The resulting software can be coupled with a wide range of biogeochemical models, as long as they conform with a rather flexible and general interface.

The main focus of this work is to describe the necessary changes to the software to port it to GPU hardware and to determine the resulting speed-up (if there is any). The motivation for this clearly is that matrix-vector multiplications are one of the numerical operations predestined for GPUs. Thus we expect a big performance gain when

GMDD
5, 2179–2214, 2012

porting just the transport and neglecting the spatially local and usual nonlinear biogeochemical interactions. We furthermore wanted to check how this performance gain behaves when incorporating the biogeochemical models. For this purpose, we take here two examples with just two tracers. One oft them is a linear uncoupled one, describing for example the radioactive decay of two species. The second one is the coupled and nonlinear N-DOP model described by Parekh et al. (2005) and used, for example, in Kriest et al. (2010). High performance computing on GPU or other special, highly parallel hardware becomes more and more attractive also in climate and geophysical research, we just refer to NVIDIA Corporation (2012, docs/IO/43499/Headwave.pdf) and Hanappe et al. (2011); Horn (2012) as some examples here. To our knowledge there is no publication about using GPU for marine ecosystem simulations.

Since we want to explicitly show what steps were necessary for the mentioned CPU-to-GPU port, we start be describing the original software for the ecosystem spin-up and the used ecosystem models in Sect. 2. Afterwards we describe the standards, tools and libraries used for GPU programming in Sect. 3. We show which GPU-adapted software could be used and what kind of adaption we additionally had to make ourselves. This is topic of Sect. 4. Then we show the numerical results for the two models, both on CPU and GPU hardware. We end the paper with some conclusions and an outlook, especially treating the option of a nested parallel simulation using

2 Coupled marine tracer transport simulation using transport matrices

A marine ecosystem is usually modeled as a system of equations for the ocean circulation and the transport of temperature, salinity and the incorporated biogeochemical tracers including their interactions. A fully coupled simulation – reflecting the fact that tracers are advected by the ocean circulation, their diffusion is dominated by the turbulent mixing of marine water and vice versa a tracer concentration may effect the ocean circulation – is computationally expensive. Such a coupled (also called “online”) simulation in three space dimensions can be performed on high-performance computers

GMDD
5, 2179–2214, 2012

only, even more if steady annual cycles – whose simulation requires long-term spin-ups – are looked for.

In contrast, a so-called "offline" computation is a simplified approach for tracers that are (or are regarded as) "passive", i.e. they do not affect the ocean physics, or this influence is neglected. This results in a one-way coupling from the ocean circulation to the tracer dynamics only, where the pre-computed circulation data (advection velocity vector field $\mathbf{v}$, mixing coefficient κ, temperature, and optionally salinity) enter the tracer transport equations as forcing.

Here $(\boldsymbol{x}, t)$ denotes a point in the space-time cylinder $\Omega \times [0,T]$ with $\Omega \in \mathbb{R}^3$ being the spatial domain (i.e. the ocean) with boundary $\Gamma = \partial\Omega$ and $[0,T], T > 0$, the time interval.

With this data given, a marine ecosystem model considered in an offline computation consists of the following system of parabolic partial differential equations (here for n tracers y_i summarized in the vector $\boldsymbol{y} = (y_i)_{i=1,\ldots,n}$):

$$\frac{\partial y_i}{\partial t} = \nabla \cdot (\kappa \nabla y_i) - \nabla \cdot (\mathbf{v} y_i) + q_i(\boldsymbol{y}), \quad i = 1, \ldots, n, \tag{1}$$

in the space-time cylinder $\Omega \times [0,T]$ with $\Omega \in \mathbb{R}^3$ being the spatial domain (i.e. the ocean) and $[0,T], T > 0$, the time interval. Additionally, homogeneous Neumann boundary conditions on $\Gamma = \partial\Omega$ for all tracers y_i are imposed. The source-minus-sink or coupling terms q_i in general are nonlinear and represent growth, dying, and tracer interaction. Each of them not necessarily depends on *all* tracers in $\boldsymbol{y}$, but usually on more than the y_i itself, thus reflecting tracer coupling given for example, by grazing etc. Here, we neglect the additional dependency on the space and time coordinates $(\boldsymbol{x}, t)$ in the notation for brevity. The q_i also include model parameters (as growth and dying rates, sinking velocities etc.) that are often subject to identification or estimation. They are usually spatially and temporally constant and not mentioned explicitly here.

GMDD
5, 2179–2214, 2012

2.1 Transport matrices

Since in an offline simulation the ocean circulation data is only used as pre-computed input for the tracer transport equations (Eq. 1), the spatial differential operators therein can be represented as a linear operator and the equations can be formally written as

$$\frac{\partial y_i}{\partial t} = L(\kappa, \mathbf{v}, t) y_i + q_i(\boldsymbol{y}), \quad i = 1, \ldots, n. \tag{2}$$

Here, $L(\kappa, \mathbf{v}, t)$ is a linear operator comprising the whole transport, i.e. diffusion and advection, for the given ocean circulation data κ and $\mathbf{v}$. It is time-dependent since the circulation date depend on time also, no matter if a transient computation shall be performed or a stable annual cycle with climatological input data is looked for. The operator A is identical for all tracers if the molecular diffusion of the tracers is small compared to the turbulent mixing, which is a reasonable simplification.

The idea of the "Transport Matrix Method (TMM)" introduced in Khatiwala et al. (2005) is to compute or approximate the matrices that represent an appropriate discretization of L. This is done by running time-steps of the ocean model that has produced the circulation data $\mathbf{v}, \kappa$ etc., with special, only locally non-zero initial distributions for one tracer. By varying the support of the initial distributions over the whole spatial domain, an approximation for one or several time steps can be obtained, which can be then used to build up a matrix representation of L. A comprehensive discussion of the temporal and spatial discretization as well as the process of evaluating transport matrices, especially in combination with operator splitting schemes can be found in Khatiwala et al. (2005). For our results we used twelve implicit and twelve explicit transport matrices, which represent monthly averaged diffusion and advection. The matrices are interpolated linearly to the corresponding discrete time step during simulation.

As a result, we obtain the following fully (temporal and spatial) discrete scheme where we now denote by $\boldsymbol{y}_j$ the appropriately arranged vector of the values of all n tracers on all spatial grid points at time step j. In the same way, we denote by $\boldsymbol{q}_j$ the vector of the discretized source-minus-sink terms at all spatial grid points in time step

GMDD
5, 2179–2214, 2012

j. Using the TMM for a fixed time step-size τ, the time integration scheme for (Eq. 2) reads

$$\boldsymbol{y}_{j+1} = A_{\mathrm{imp},j}(A_{\mathrm{exp},j}\boldsymbol{y}_j + \tau \boldsymbol{q}_j(\boldsymbol{y}_j)) =: \varphi_j(\boldsymbol{y}_j), \qquad j = 0, \ldots, n_\tau - 1. \tag{3}$$

Here n_τ is the total number of time steps and $A_{\mathrm{imp},j}, A_{\mathrm{exp},j}$ are the implicit and explicit transport matrices at time step j. The matrices are block-diagonal and sparse and depend on the used time-stepping scheme: if – as a simple and not realistic example – the whole system would be solved explicitly by an Euler step, $A_{\mathrm{imp},j}$ would be the identity and $A_{\mathrm{exp},j}$ would be the discrete counterpart of $I + \tau L(\kappa, \mathbf{v}, t_j)$. Summarizing, starting from a vector $\boldsymbol{y}_0$ of initial values, each step in the time integration scheme (Eq. 3) to solve the tracer transport equations (Eq. 1) consists of the evaluation of the source-minus-sink term and two matrix-vector multiplications.

Table 1 shows typical values for the sizes and sparsity of transport matrices generated by the MITgcm ocean model for two typical spatial resolutions, see Khatiwala et al. (2005); Piwonski and Slawig (2011). Figure 1 shows the sparsity patterns.

2.2 Applying parallel algorithms using the PETSc library

From the sizes of the transport matrices it can be deduced that a parallelization of the matrix-vector multiplication occurring every time step can significantly speed up the process, even more when computing the steady annual cycle by the pseudo-time stepping (or fixed point iteration) described above. In the CPU setting, see for example Piwonski and Slawig (2011), the parallelization is carried out on a distributed memory architecture using the "Message Passing Interface (MPI)" standard (Argonne Nat. Lab., 2012). In our case this means that every matrix-vector multiplication is distributed among several processor cores. In order to avoid the direct implementation of MPI directives, we make use of the PETSc library (Balay et al., 2012). It is a collection of data structures and algorithms for the parallel solution of numerical problems and provides interfaces (APIs) to programming languages as Fortran, C, C++, Python, and

GMDD
5, 2179–2214, 2012

MATLAB®. Main advantages of PETSc for our application are the hiding of the MPI directives in the matrix-vector-multiplication routines and the usage of an efficient sparse matrix storage format, in our case the default PETSc format, namely the "AIJ" or "Yale sparse" or "CSR (compressed sparse row)" format. Since the amount of available storage on a GPU is limited, the storage requirements of the matrices are essential. We thus give a short description of the CSR format as a C structure:

```
typedef struct {
    unsigned int nnz;
    unsigned int nrows;
    unsigned int ncols;
    double *Vals;
    int *ColInd;
    int *RowPtr;
} CSRMat;
```

The format only stores non-zero values and needs as storage

$$\texttt{nnz} \cdot \text{sizeof}(\texttt{double}) + (\texttt{nnz} + \texttt{nrows} + 4) \cdot \text{sizeof}(\texttt{int}) \tag{4}$$

for a matrix with entries in double precision. Here `nnz`, `nrows` refer to the values in Table 1. The sparsity patterns (represented by the values `nnz`, `nrows`, `ncols` and the arrays `ColInd` and `RowPtr`) are the same for $A_{\text{imp},j}$ in all time steps (i.e. for all j) on one hand and for all $A_{\text{exp},j}$ on the other. Thus it is also possible to store only two sparsity patterns for the whole set of matrices, which might be useful when the memory on the GPU is too small to store the complete CSR data structures for each of them.

2.3 Computation of steady annual cycles

Computing a periodic solution of the discretized system (Eq. 3) means looking for a fixed point of the mapping $\Phi = \varphi_{n_\tau - 1} \circ \cdots \circ \varphi_0$, i.e. for a trajectory $(\boldsymbol{y}_j)_{j=0,\ldots,n_\tau}$ with

GMDD
5, 2179–2214, 2012

$$\boldsymbol{y}_{n_\tau} = \Phi(\boldsymbol{y}_0) = \boldsymbol{y}_0. \tag{5}$$

Thus one application of the mapping Φ corresponds to the computation of one year model time (or *model year*). The time-step used in our computations was 3 h which corresponds (taking 360 days a year) to $n_\tau = 2880$. The discretization of the biogeochemical terms q_i may include shorter time-steps (typically 8 per outer 3-h step).

The whole iteration to compute a steady cycle (or fixed point) now consists of a repeated application of the mapping Φ, i.e.

$$\boldsymbol{y}^{l+1} = \Phi(\boldsymbol{y}^l, \boldsymbol{u}), \quad l = 0, \ldots, n_l - 1, \tag{6}$$

where $\boldsymbol{y}^l$ is the vector of discretized tracer after l model years, i.e. $\boldsymbol{y}^l := \boldsymbol{y}_{l \cdot n_\tau}$, and n_l the total number of model years necessary to reach a steady annual cycle. The resulting structure of the spin-up is sketched in Algorithm 1.

Algorithm 1: Marine Ecosystem Spin-up using TMM

Require Set of monthly averaged transport matrices $A_{\text{imp},j}, A_{\text{exp},j}$, initial tracer distribution $\boldsymbol{y}_0$, time step τ
Ensure At the end $\boldsymbol{y}$ is a tracer distribution (at one point in time) of a steady annual cycle
$\boldsymbol{y} = \boldsymbol{y}_0$
repeat
  for $j = 0, \ldots, n_\tau - 1$ **do**
    compute biogeochemical source-minus-sink terms: $\tilde{\boldsymbol{y}} = \boldsymbol{q}(\boldsymbol{y})$
    interpolate the monthly averaged transport matrices to the current time step j
    perform explicit step: $\hat{\boldsymbol{y}} = A_{\text{exp},j}\boldsymbol{y}$
    perform implicit step: $\boldsymbol{y} = A_{\text{imp},j}(\hat{\boldsymbol{y}} + \tau\tilde{\boldsymbol{y}})$
  end for
until steady annual cycle is reached

GMDD
5, 2179–2214, 2012

In our original implementation (Piwonski and Slawig, 2011) used on a multiprocessor CPU cluster, the biogeochemical part (line 4) is implemented in Fortran, whereas the remainder of the code is realized in C. There is a difference with respect to the access of the tracer data that becomes important later on the GPU: for the biogeochemical computations (line 4), the values of the separate tracers and also on different spatial grid points (compare Eq. 7) are needed simultaneously. In contrast, the matrix-vector products (lines 6, 7) are executed separately for each tracer, thus allowing to store and work with one block of the transport matrices only. Each matrix-vector product is computed by one call to the internally parallelized PETSc routine `MatMult()`.

For the interpolation step in line 5, three other PETSc routines are used (for explicit and implicit matrix separately) to compute the appropriately weighted matrices:

```
MatCopy(A[i_alpha], *A_work, SAME_NONZERO_PATTERN);
MatScale(*A_work, alpha);
MatAXPY(*A_work, beta, A[i_beta], SAME_NONZERO_PATTERN);
```

These three routines together compute a linear interpolant or convex combination of two succeeding monthly averaged matrices, which are stored in the array `A` starting at index `i_alpha` and `i_beta`, respectively. Thus the above lines compute

```
A_work = alpha * A[i_alpha] + beta * A[i_beta]
```

which gives the desired interpolated matrix in `Awork`, if `alpha,beta` are chosen correctly with respect to the time step j. Since these three routines (copying, scaling and adding) operate locally on every element of the matrix only, they are predestinated for an efficient parallelization, especially on the GPU hardware.

From several computations it can be observed that after about $n_I = 3000$ iterations (model years), a numerical steady solution (up to an accuracy of about 10^{-2} in discrete $L^2(\Omega)^n$ norm) is obtained. Thus we refer to this as a "converged steady annual cycle". This value of n_I was also used in (Kriest et al., 2010). The residual can be further decreased by using a higher number n_I of model years.

GMDD
5, 2179–2214, 2012

2.4 Ecosystem and biogeochemical model examples

We use two simple models to test the computational gain possible with the GPU hardware. Each of them has two tracers (i.e. $n = 2$ in Eq. 1 and thereafter). Both are described and available in Piwonski and Slawig (2011).

The first one is a simple radioactive decay model which is uncoupled and has the autonomous source-minus-sink term

$$\boldsymbol{q}(\boldsymbol{y}) = \begin{pmatrix} -\lambda_1 y_1 \\ -\lambda_2 y_2 \end{pmatrix}.$$

The parameters $\lambda_1, \lambda_2 > 0$ are the decay rates of the two radioactive elements. We chose Iodine I^{131} with $\lambda_1 \approx 44.88$ and Caesium Cs^{137} with $\lambda_2 \approx 0.0331$. This uncoupled model is used in order to test the gain in CPU time for the pure matrix-vector multiplication and interpolation in the TMM.

The second model is a typical biogeochemical model, including both coupling and nonlinearities. It is based on the N-DOP model described in Parekh et al. (2005) which was also used in Kriest et al. (2010), from which we basically take the notation. The model incorporates phosphate (nutrients, N, y_1) and dissolved organic phosphorus (DOP, y_2). The source-minus-sink term is split up into the upper, sun-lit or euphotic zone Ω_1 with depth l', and the lower, non-euphotic zone Ω_2:

$$q_1(\boldsymbol{y}) = \begin{cases} -f(y_1) + \lambda y_2 & \text{in } \Omega_1 \\ (1-\sigma)\frac{\partial}{\partial z} F(y_1) + \lambda y_2 & \text{in } \Omega_2 \end{cases}$$

$$q_2(\boldsymbol{y}) = \begin{cases} \sigma f(y_1) - \lambda y_2 & \text{in } \Omega_1 \\ -\lambda y_2 & \text{in } \Omega_2, \end{cases}$$

z being the vertical coordinate. The biological production (the net community productivity) is calculated as a function

$$f(y_1) = \alpha \frac{y_1}{y_1 + K_N} \frac{I}{I + K_I}$$

GMDD
5, 2179–2214, 2012

of nutrients y_1 and light I. The dependence on the latter is omitted here in the notation for brevity. The production is limited by a half saturation function, also known as Michaelis-Menten kinetics, and a maximum production rate parameter α. Light is modeled as a portion of short wave radiation I_{SWR}, which is computed as a function of latitude and season following the astronomical formula of Paltridge and Platt (1976). The portion depends on the photo-synthetically available radiation σ_{PAR}, the ice cover σ_{ice}, and the exponential attenuation of water, i.e.

$$I = I_{SWR}\sigma_{PAR}(1 - \sigma_{ice})\exp(-zK_{H_2O}).$$

A fraction σ of the biological production remains suspended in the water column as dissolved organic phosphorus, which remineralizes with a rate λ. The remainder of the production sinks as particulate to depth where it is remineralized according to the empirical power law relationship determined by Martin et al. (1987),

$$F(y_1) = \left(\frac{z}{I'}\right)^{-b} \int_0^{I'} f(y_1)\mathrm{d}z. \quad (7)$$

Similar modeling of biological production can be found for example in Dutkiewicz et al. (2005). The remaining model parameters are given in Table 2.

3 GPU computing with CUDA

In this section we describe the basic architecture of GPUs and the consequences on exploiting their advantages for scientific computing, and give an overview of some useful libraries. We concentrate on the "CUDA" (NVIDIA Corporation, 2012) programming framework. One alternative is, for example, "OpenCL" (The Khronos Group, 2012).

NVIDIA as one of the leading producers of graphic cards has developed its own "Compute Unified Device Architecture (CUDA)" as a parallel architecture for executing computational expensive code on GPUs. By exploiting the architecture of graphics

GMDD
5, 2179–2214, 2012

cards as well as the increased memory bandwidth, it is possible to perform a far greater number of floating point operations per second (FLOPS) than on CPUs. While CPUs have about one to eight cores each with up to 4 GHz clock rate, GPUs nowadays do have a lower clock rate, but hundreds of cores which can run multiple threads simultaneously.

The basic unit of the CUDA architecture is called "kernel". A kernel is a piece of program code executed on the graphics card many times in parallel threads. These threads are addressed via a so-called "grid" of blocks. A "block" consists of threads, and its size is determined by a size with up to three dimensions. up to specification. By the CUDA language construct

```
kernel<<<gridSize, blockSize>>>()
```

the blocks are created and run in parallel by the GPU. The maximum number of threads equals the product of block size and number of blocks.

The GPU consists of several so-called "Streaming Multiprocessors (SM)". Each SM has its own buffer memory, registers, and a number of cores. The cores have their own calculators for integer and floating-point calculation. For example, the GeForce GTX 480 used here has 15 SM with each 32 cores, i.e. a total of 480 cores. On a core the smallest executable unit is a so-called "warp", which consists of 32 threads. The total number of threads that can run simultaneously on a multiprocessor is dependent on NVIDIA's so-called "Compute Capability (CC)" of the graphics chips. For the GTX 480 the limit is 1536 threads (p. 159 NVIDIA Corporation, 2011) which results in a maximum number of concurrent threads for the entire GPU of $15 \times 1536 = 23040$.

The device memory on the GPU is divided into three types of physical and virtual portions. At first, a thread has access to its own private memory (which is, depending on the CC between 16 kB and 512 kB, see NVIDIA Corporation, 2011, p. 159). Secondly, threads within one block have access to a shared memory of between 16 kB to 48 kB. Finally, all threads have access to a shared global memory whose size is limited by the total amount of memory of the GPU.

GMDD
5, 2179–2214, 2012

While working with CUDA, it is essential to distinguish between the memories of CPU and GPU. In order to run kernel code on the GPU, all data must be transferred from the host memory of the CPU to the device memory on the GPU.

To interact with the GPU via CUDA, a program consists of two components: first, the kernel code consisting of the CUDA Instruction Set (called PTX), on the other, the C code which calls this kernel. NVIDIA provides a compiler (`nvcc`) that translates C code in PTX code, and that behaves similarly to the C compiler in the GNU compiler Collection `gcc`. The extension for CUDA source code files is `.cu`. NVIDIA also provides a port of the GNU debugger `gbd`.

3.1 Libraries

We use some libraries that simplify several basic actions and algorithm while working with GPUs. The first one is "Thrust" (Thrust, 2012), a C++ collection of generic algorithms, similar to the C++ "Standard Template Library STL", that exploit the parallelism of the GPU in a transparent way. Using Thrust, many problems can be solved without even writing code for the GPU. We here show an example operation: on the host system a large array is created using random numbers, which then is ported to the GPU, sorted there, and transferred back to the CPU:

```
int main(void) {

  // generate random numbers on the host
  thrust::host vector<int> h_vec(1 << 24);
  thrust::generate(h vec.begin(), h_vec.end(), rand);
  // transfer data to the device
  thrust::device vector<int> d_vec = h_vec;
  // sort data on the device
  thrust::sort(d_vec.begin(), d_vec.end());
  // transfer data back to host
  thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());
```


GMDD
5, 2179–2214, 2012

```
    return 0;
}
```

The second library is “Cusp”, which provides data types for sparse matrices and algorithms for basic linear algebra operations on them. The following example shows how a matrix in the “HYB” sparse storage format is generated on the device (i.e. the GPU) memory, how it is filled with values from a file, and how then a linear system is solved by the “Conjugate Gradient” method on the GPU.

```
int main(void) {
    // create an empty sparse matrix structure (HYB format)
    cusp::hyb_matrix<int, float, cusp::device_memory> A;
    // load a˜matrix stored in MatrixMarket format
    cusp::io::read_matrix_market_file(A, "filename.mtx");
    // allocate storage for solution x and right hand side b
    cusp::array1d<float, cusp::device_memory> x(A.num_rows, 0);
    cusp::array1d<float, cusp::device_memory> b(A.num_rows, 1);
    // solve  A˜x = b with Conjugate Gradient method
    cusp::krylov::cg(A, x, b);
    return 0;
}
```

As can be seen, all data structures in Cusp have a parameter that determines whether it is stored in CPU or GPU memory. All operations on the data will then take place in the respective storage area. For our application, in particular the structure `cusp::csr_matrix` for the CSR format and the matrix-vector multiplication routine `cusp::multiply` that uses the algorithm described in Bell and Garland (2008, 2009), which was specially developed for GPUs, are important.

The third library we used was the preliminary implementation of PETSc for the CUDA architecture presented in Minden et al. (2010). With the help of the Thrust and Cusp libraries, a large part of the PETSc `Vector` and some parts of the `Matrix` class have

GMDD
5, 2179–2214, 2012

been implemented. The fundamental problems of interaction of PETSc with the GPU have been resolved, but only the routines that were necessary for the example treated in Minden et al. (2010) have been implemented. Basically this "PETSc GPU" extends the built-in structures by a value that indicates in which memory the most recent data are stored. This guarantees that the correct data are available (and if necessary copied to) the memory that is currently used.

There is a recent PETSc version 3.3 which "supports CUDA" (Balay et al., 2012), but this version was not used in this work and we did not study if features equivalent to our extensions have already been included.

4 Port of the marine ecosystem simulation onto the GPU

We now describe the necessary modifications and extensions of the original program that was running on a multiprocessor CPU cluster in order to run the simulation on a GPU. Basically these modifications are extensions of PETSc GPU, modifications necessary to use of a CUDA Fortran compiler for the biogeochemical model parts and some conversions between different data structures.

4.1 Necessary extensions of PETSc GPU

The preliminary PETSc GPU implementation was designed to solve systems of equations, and thus not all functions necessary for our applications were included. To avoid any copying of data between CPU and GPU storage that would have destroyed the speed-up, we thus had to extend the library. In our case, the three PETSc routines `MatCopy`, `MatScale`, and `MatAXPY` mentioned in Sect. 2.3 had to be modified.

PETSc functions in C have to conform to a format which basically looks like this:

```
PETScErrorCode DoSomething(...) {

    //  function statements
```


GMDD
5, 2179–2214, 2012

```
        PETScFunctionReturn(0);
    }
```

Here the dots denote the parameter list. PETSc requires each function to return a `PETScErrorCode` that describes its success or failure and is returns by the call to `PETScFunctionReturn()` in the last line.

The PETSc wrapper function

```
MatCopy(Ain, *Aout, SAME_NONZERO_PATTERN);
```

copies the values of matrix `Ain` into matrix `Aout`. The third parameter indicates if both matrices have the same sparsity pattern, which then avoids unnecessary memory allocations.

For our GPU version `MatCopy_SeqAIJCUSP()` for the "AIJ" sparse format, it is theoretically possible for both matrices that are either currently in the GPU memory, the CPU memory or in both. For a complete and correct implementation, it would have been necessary to cover all these cases, and accordingly select memory in which the matrices are actually copied. For our application it was sufficient to cover only the case where the matrices are both in the GPU memory, thus only this case was implemented. Therefore, an additionally implemented function `MatCUSPCopyToGPU()` ensures that both matrices are in the GPU memory. Here we could make use of available Cusp functions.

Both PETSc functions `MatScale()` and `MatAXPY()` implement typical linear algebra subproblems. As a consequence, they could be nearly completely realized using the Cusp BLAS library on the GPU. The operations are only performed on the non-zero matrix elements.

4.2 PGI CUDA-Fortran

Many biogeochemical models are implemented in Fortran. Since we want to port them to the GPU, we need a Fortran compiler that generates code that can be linked against the C code compiled by the `nvcc` compiler such that the resulting executable runs on

GMDD
5, 2179–2214, 2012

the GPU. At the time of this work there was one candidate, namely the "PGI CUDA Fortran compiler" (The Portland Group, 2012). It extends the language by constructs for calling kernel functions as well as the CUDA API functions. The syntax of a kernel call in Fortran is

```
call kernel <<<...>>>
```

and thus similar to CUDA C++ . There are some extensions compared to CUDA C++, but also some restrictions. For details we refer to the manual (The Portland Group, 2011a, p. 14).

4.3 Other extensions to the implementation on the CPU

As described in Sect. 2.3, there are two different data structures useful for the spin-up using the TMM (see Algorithm 1): one for the biogeochemical source-minus sink terms, where all tracers are needed and thus kept together in one structure, and another one for the multiplication with the transport matrices, where every tracer is kept separately to reduce the storage requirements for the matrices. Thus a copying between these two data structures is necessary in every step of the algorithm. For the use on the GPU, three copying functions in the original code were modified using the Thrust library.

4.4 The compilation process for the GPU

Here we briefly sketch the overall compilation and linking process of the resulting code for the use on the GPU. The process is visualized in Fig. 2.

In a first step (top right in Fig. 2), the biogeochemical model implemented in Fortran in the file `model.F`, together with a driver routine `driver.CUF`, is translated by the PGI CUDA-Fortran pre-compiler `pgcpp` into a form (`driver_model.CUF`) that can be coupled to the remaining part of the simulation package written in C. The Fortran compiler `pgfortran` then generates an object file `driver_model.o`.

The driver routine `driver.CUF` has two tasks: at first the Fortran compiler requires that all functions which shall run on the GPU are marked with the `device` attribute, see

GMDD
5, 2179–2214, 2012

The Portland Group (2011b). Since the compiler has no ability to set default attributes for all functions, it is necessary to integrate them through a pre compiler definition. Therein the Fortran keyword `subroutine` is replaced by

```
attributes(device) subroutine
```

Secondly, the driver provides support functions for the three entry points into the biogeochemical model, namely (i) the evaluation of the source-minus-sink terms, (ii) the initialization and (iii) de-initialization functions of the model. These three functions need corresponding kernel functions on the GPU.

On the other hand (top left of Fig. 2), the original code of the C files is translated as in the CPU version with the MPI wrapper of the GNU C compiler `mpicc`, while the extensions in form of CUDA kernels and their wrappers are translated with the CUDA compiler `nvcc`.

Finally all object code is linked against some additional PGI Fortran libraries, which gives the final executable file (here called `metos3d.exe`).

5 Numerical results

In this section we compare the spin-up performed on CPU and GPU hardware. We use the two models described in Sect. 2.4. A special emphasis lies on the time needed for the different parts of the spin-up, namely the matrix-vector multiplications, the evaluation of the biogeochemical source minus-sink terms, and the matrix interpolation which includes matrix copying.

5.1 Setup

The test system for the GPU is described in Table 3. The used GPU, the GeForce GTX 480, is a consumer card and already an older model of the GTX series. As of this writing, there are already successor models with more cores. Moreover, the NVIDIA "Tesla" series is especially designed for numerical computations on GPU hardware.

GMDD
5, 2179–2214, 2012

For the following test setups and results, only one CPU and one graphics card was used. When reproducing the tests it is crucial to ensure that the graphic card is not used by any "real" graphic output by the operating system. Many graphical user interfaces are now GPU-accelerated, so the results can be falsified. In particular, it is possible that there is not enough memory on the card for the used monthly set of (then totally 24) implicit and explicit transport matrices. When the 2.8125° horizontal resolution is used, they need about 1 GB memory. A monthly averaged set of transport matrices based on a 1° resolution is too big for the used GPU system.

To perform an accurate time measurement, the profiling system of PETSc was used, see Balay et al. (2012, docs/manualpages/Profiling/index.html). All measurements are means of 100 runs. No further code optimization was performed on the code for the GPU, such that all comparisons are somehow "fair".

5.2 Results

We start by comparing the time needed for one annual cycle (i.e. one model year) in the spin-up for both the I-Cs and the N-DOP model. The reduction factor is about 12 for the simpler I-Cs model and about 22 for the more complex N-DOP model.

Using the profiler, we evaluated the time needed for the different steps in the "repeat-until" loop of Algorithm 1 corresponding to one annual cycle. The respective functions are listed in Table 5. Note that for the I-Cs model `BGCStep` is just a scaling of the tracer vector.

Figures 3 and 4 show the results for both models with a block size of 160 on the GPU. We discuss the effect of different block sizes later on. We make the following observations. For the uncoupled I-Cs model (without tracer coupling) we find:

- The effort needed on both CPU and GPU for `BGCStep` is negligible.
- On the CPU, the effort for the three functions used for the matrix interpolation are expensive compared to the matrix multiplications in `MatMult`, which perform the tracer transport, themselves. On the CPU it would be thus much more efficient to

GMDD
5, 2179–2214, 2012

avoid the interpolation, but this would result is a huge storage effort since then matrices for every time-step would have to be stored.

- Since the operations needed for the interpolation are very appropriate for the parallelization on the GPU, they become negligible there, such that the main effort now is the matrix-vector multiplication.
- The matrix-vector multiplication is sped up by a factor of 4.5 from CPU to GPU.

For the N-DOP model that includes nonlinear and also non-local tracer coupling we can see:

- More than two third of computational effort on the CPU is spent for `BGCStep`. The fractions of the remaining time spent on the linear algebra operations correspond to those for the I-Cs model: again the interpolation takes significantly more time than the matrix-vector-multiplication in `MatMult`, i.e. the tracer transport itself.
- On the GPU, the overall time needed is nearly equally split between `BGCStep` and `MatMult`.
- For this model, the speed-up from CPU to GPU is again 4.5 for `MatMult` (which is natural, since this operation is the same for both models).
- For `BGCStep`, the speed-up is 34.2, which is rather surprising since the nonlinear and coupled operations at first seem not to be predestined for the GPU.

Another parameter to be examined is the block size for the PGI Fortran CUDA kernel call of the biogeochemical model. The block size describes the number of vertical profiles that are processed within a block. Figure 5 depicts the time needed on the GPU depending on the block size. In both models, there are strong fluctuations up to 100 % of the needed time as a function of block size. In both models the absolute minimum of time (I-Cs: 0.35 s, N-DOP: 13 s) is obtained for a block size of 160. That is the reason that this size was used in the above comparisons. Furthermore, both graphs show

GMDD
5, 2179–2214, 2012

similar occurrence of minima and maxima. For the N-DOP model these extrema are even bigger, since the part of the biogeochemical model in the total computation time is significantly higher.

As a last comparison, we tested the performance of the GPU against CPU clusters with three more or less current processor types. As can be seen in Fig. 6, the single (!) GPU for the N-DOP model simulation has the same performance as approximately 56 Barcelona, 28 Westmere, and 17 Gainestown processors.

6 Conclusions

In order to port the existing implementation of the spin-up of marine ecosystem models using the Transport Matrix Method from CPU to GPU hardware, some extensions of necessary libraries had to be done. These require some knowledge in the computing architecture of the used CUDA programming framework. On the other hand, several libraries for the GPU are already available, and since GPU computation is a growing field of research and applications, there will be more in the future. In order to compile Fortran code for the GPU, a commercial compiler was necessary.

Concerning the computational gain, in our computations the effect was totally as expected (about 20). Overall, the used single consumer (and even not latest) GPU outperforms a considerable high number of CPUs in a cluster architecture. The acceleration effect on the matrix-vector multiplications was poorer (about 4.5) than expected, whereas the effect on simple matrix copying, scaling and addition was impressive. As a surprise, also the effect on the computation of the nonlinear, coupled and also non-local source-minus-sink terms in one of the tested biogeochemical models was quite high (about 34). The considered biogeochemical models were still rather simple, but because of this last point it would be very interesting how the spin-up performs on a GPU for more complex models. Additionally, also tests with more powerful GPUs would be a next step. And, last but not least, also the coupling of two or more GPUs

GMDD
5, 2179–2214, 2012

would be interesting, in order to observe if the then necessary data transfer slows down the whole computation and thus reduces the performance gain.

Acknowledgements. The GPU computations were performed with kind permission and support of the Research Group for Communication Systems at Christian-Albrechts-Universität zu Kiel, Institute for Computer Science. The access to the HLRN computing facility was kindly provided by the group Marine Biogeochemical Modelling at IfM GEOMAR, Kiel, Germany.

References

Argonne Nat. Lab.: The Message Passing Interface (MPI) standard, available at: http://www.mcs.anl.gov/research/projects/mpi/ (last access: 19 July 2012), 2012. 2185

Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H.: PETSc web page, available at: http://www.mcs.anl.gov/petsc (last access: 19 July 2012), 2012. 2185, 2194, 2198

Bell, N. and Garland, M.: Efficient sparse matrix-vector multiplication on CUDA, Technical Report NVR-2008-04, NVIDIA Corporation, Santa Clara, CA 95050, USA, 2008. 2193

Bell, N. and Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors, in: SC 2009: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, New York 2009, 1–11, ACM, 2009. 2193

Dutkiewicz, S., Follows, M., and Parekh, P.: Interactions of the iron and phosphorus cycles: a three-dimensional model study, Global Biogeochem. Cy., 19, 1–22, 2005. 2190

Hanappe, P., Beurivé, A., Laguzet, F., Steels, L., Bellouin, N., Boucher, O., Yamazaki, Y. H., Aina, T., and Allen, M.: FAMOUS, faster: using parallel computing techniques to accelerate the FAMOUS/HadCM3 climate model with a focus on the radiative transfer algorithm, Geosci. Model Dev., 4, 835–844, doi:10.5194/gmd-4-835-2011, 2011. 2182

Horn, S.: ASAMgpu V1.0 – a moist fully compressible atmospheric model using graphics processing units (GPUs), Geosci. Model Dev., 5, 345–353, doi:10.5194/gmd-5-345-2012, 2012. 2182

Khatiwala, S.: A computational framework for simulation of biogeochemical tracers in the ocean, Global Biogeochem. Cy., 21, GB3001, doi:10.1029/2007GB002923, 2007. 2181

Khatiwala, S., Visbeck, M., and Cane, M.: Accelerated simulation of passive tracers in ocean circulation models, Ocean Model., 9, 51–69, 2005. 2184, 2185

GMDD

5, 2179–2214, 2012

Kriest, I., Khatiwala, S., and Oschlies, A.: Towards an assessment of simple global marine biogeochemical models of different complexity, Prog. Oceanogr., 86, 337–360, doi:10.1016/j.pocean.2010.05.002, 2010. 2182, 2188, 2189

Martin, J. H., Knauer, G. A., Karl, D. M., and Broenkow, W. W.: VERTEX: carbon cycling in the Northeast Pacific, Deep-Sea Res. Pt. I, 34, 267–285, 1987. 2190

Minden, V., Smith, B., and Knepley, M.: Preliminary implementation of PETSc using GPUs, in: Proceedings of the 2010 International Workshop of GPU Solutions to Multiscale Problems in Science and Engineering, Harbin, November 2010, Springer, New York, USA, 2010. 2193, 2194

M.I.T.: M.I.T. General Circulation Model, available at: http://mitgcm.org/ (last access: 19 July 2012), 2012. 2181

NVIDIA Corporation: CUDA C Programming Guide, available at: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf (last access: 19 July 2012), 2011. 2191

NVIDIA Corporation: CUDA, available at: http://www.nvidia.com/object/cuda_home_new.html (last access: 19 July 2012), 2012. 2182, 2190

Paltridge, G. W. and Platt, C. M. R.: Radiative Processes in Meteorology and Climatology, Elsevier, New York, 1976. 2190

Parekh, P., Follows, M. J., and Boyle, E. A.: Decoupling iron and phosphate in the global ocean, Global Biogeochem. Cy., 19, GB2020, doi:10.1029/2004GB002280, 2005. 2182, 2189

Piwonski, J. and Slawig, T.: Metos3D: A Marine Ecosystem Toolkit for Optimization and Simulation, CAU Kiel, Institut für Informatik, available at: http://www.informatik.uni-kiel.de/co2/software/metos3d (last access: 19 July 2012), 2011. 2185, 2188, 2189

The Khronos Group: OpenCL – The open standard for parallel programming of heterogeneous systems, available at: http://www.khronos.org/opencl/ (last access: 19 July 2012), 2012. 2190

The Portland Group: PGI Fortran Compiler Reference Manual, available at: http://www.pgroup.com/doc/pgiref.pdf (last access: 19 July 2012), 2011a. 2196

The Portland Group: CUDA Fortran Programming Guide and Reference, available at: http://www.pgroup.com/doc/pgicudafortug.pdf (last access: 19 July 2012), 2011b. 2197

The Portland Group: PGI CUDA-Fortran Compiler, available at: http://www.pgroup.com/resources/cudafortran.htm (last access: 19 July 2012), 2012. 2196

GMDD
5, 2179–2214, 2012

Thrust: Thrust, http://docs.thrust.googlecode.com/hg/modules.html (last access: 19 July 2012), 2012. 2192

GMDD
5, 2179–2214, 2012

Table 1. Resolution, sizes and sparsity of one block of the explicit and implicit transport matrices for two resolutions computed with the MITgcm.

horizontal resolution	vertical layers	matrix size (nrows)	number of non-zeros, total (nnz) and percent	
			A_{exp}	A_{imp}
2.125°	15	52 749	5 407 405 (0.1943 %)	672 779 (0.0024 %)
1°	23	682 604	76 567 216 (0.0164 %)	13 339 210 (0.0029 %)

GMDD
5, 2179–2214, 2012

Table 2. Parameters in the N-DOP model.

Name	Description	Unit
λ	remineralization rate of DOP	d^{-1}
α	maximum community production rate	d^{-1}
σ	fraction of DOP	1
K_N	half saturation constant of N	mmol P m^{-3}
K_I	half saturation constant of light	W m^{-2}
K_{H_2O}	attenuation of water	m^{-1}
b	sinking velocity exponent	1

GMDD
5, 2179–2214, 2012

Table 3. Specification of the used GPU system. Only one of the two GPUs was used.

Component	Details
CPU	2x Intel® Xeon® E5520 @2.27 GHz, each 4 hyper-threaded cores
RAM	40 GB
GPU	2x GeForce GTX 480, each with 1.5 GB RAM, 15 multi-processors, and totally 480 CUDA cores

GMDD
5, 2179–2214, 2012

Table 4. Computation time and reduction for one model year. CUDA block size: 160.

model	CPU	GPU	relation CPU : GPU
I-Cs	184.00 s	15.55 s	11.8
N-DOP	616.00 s	27.78 s	22.2

GMDD
5, 2179–2214, 2012

Table 5. The three main portions in every time step of the spin-up.

lines in Alg. 1	Function	Description
4	`BGCStep`	evaluation of source-minus-sink terms
5	`MatScale`, `MatAXPY`, `MatCopy`	interpolation of transport matrices
6, 7	`MatMult`	multiplication of transport matrices with tracer vectors

GMDD
5, 2179–2214, 2012

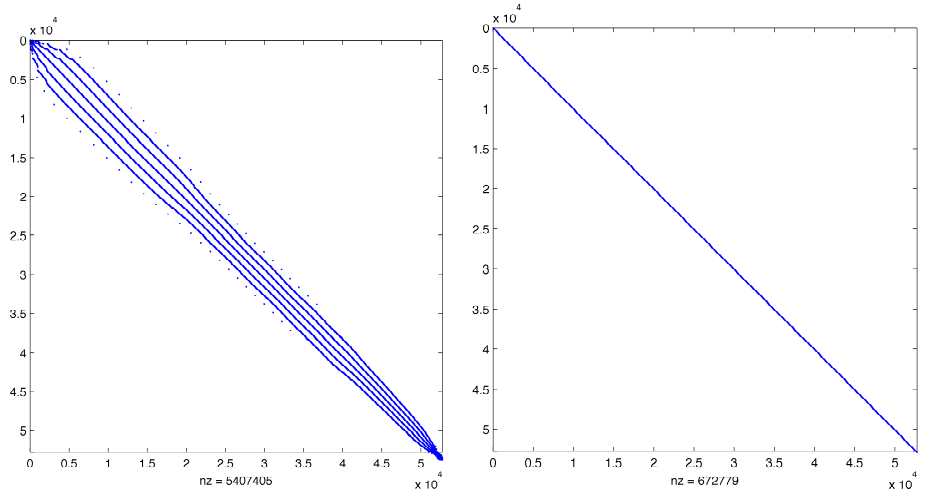

Fig. 1. One block of the explicit (left) and implicit transport matrices $A_{\mathrm{exp},j}, A_{\mathrm{imp},j}$ computed using the MITgcm model for a 2.8125° resolution (output of MATLAB®'s spy command).

GMDD
5, 2179–2214, 2012

Porting marine ecosystem model spin-up to GPUs

E. Siewertsen et al.

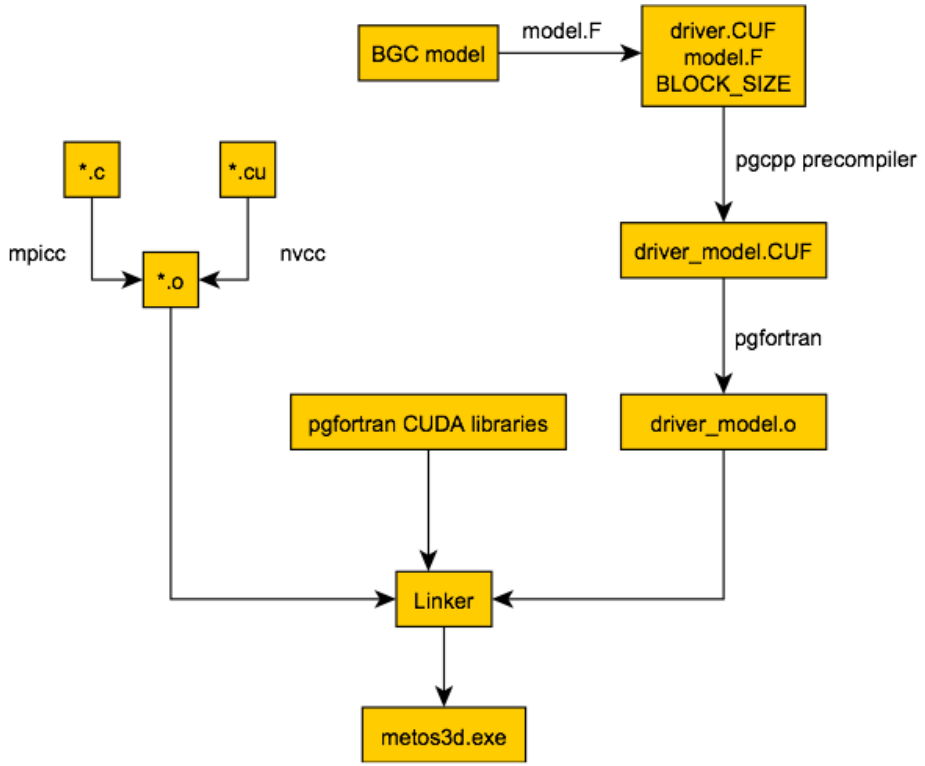

Fig. 2. Compilation and linking process of the spin-up (Algorithm 1) for usage on the GPU.

GMDD
5, 2179–2214, 2012

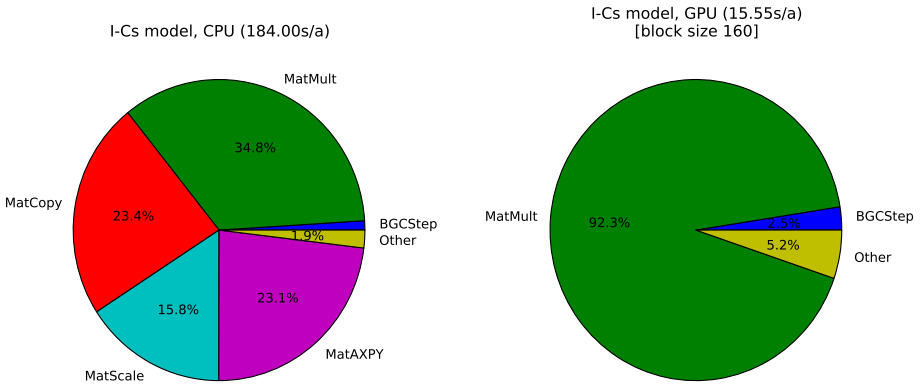

Fig. 3. Fraction of computational time needed for the different steps in one year of the spin-up (Algorithm 1) for I-Cs model on CPU (left) and GPU.

GMDD
5, 2179–2214, 2012

Porting marine ecosystem model spin-up to GPUs

E. Siewertsen et al.

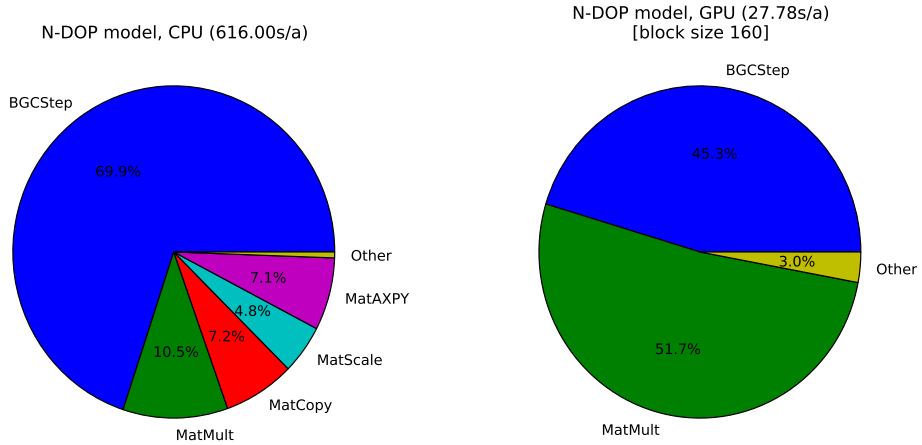

Fig. 4. Same as Fig. 3, but for the N-DOP model.

GMDD
5, 2179–2214, 2012

Porting marine ecosystem model spin-up to GPUs

E. Siewertsen et al.

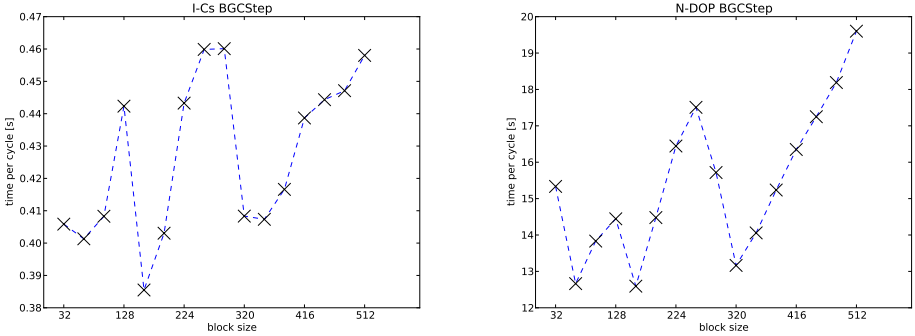

Fig. 5. Computational time needed in one year model time as a function of block size, for I-Cs (left) and N-DOP model.

GMDD
5, 2179–2214, 2012

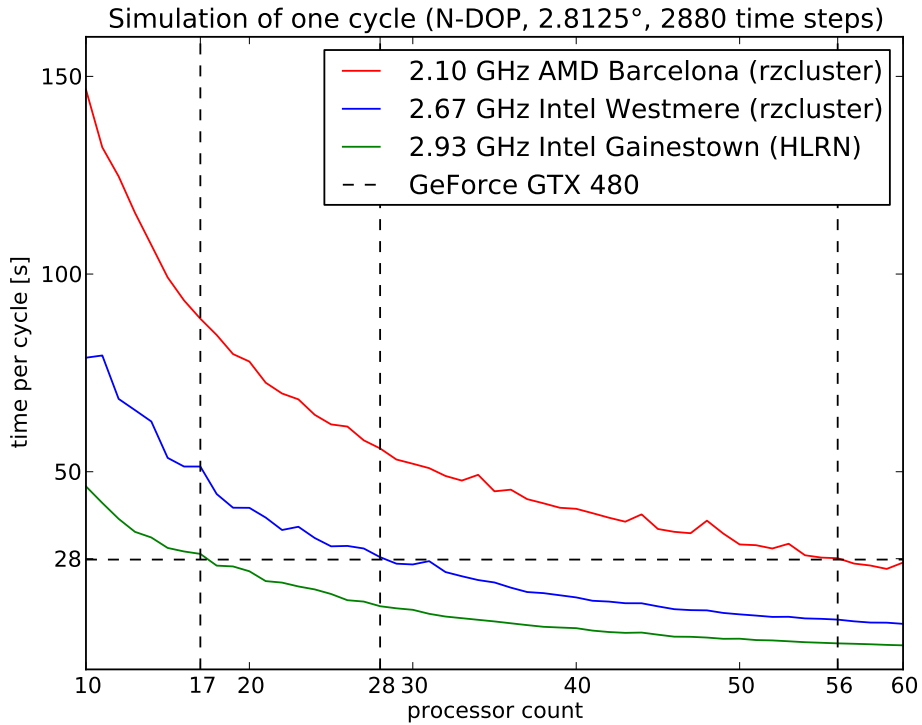

Fig. 6. Comparison between CPU cluster and the used GPU for one model year for the N-DOP model, ("rzcluster" refers to the Kiel University cluster, "HLRN" to the cluster of the *North-German Supercomputing Alliance*).

GMDD
5, 2179–2214, 2012