Porting Marine Ecosystem Model Spin-Up Using Transport Matrices to GPUs

E. Siewertsen, **J. Piwonski**, T. Slawig

CAU - Christian-Albrechts-Universität zu Kiel

20. März 2013

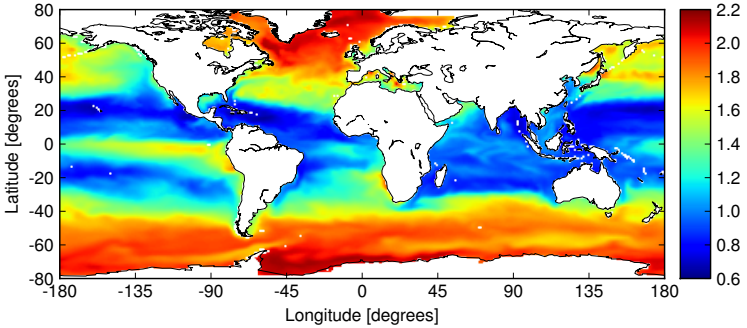
future ocean
KIEL MARINE SCIENCES

Outline

- Motivation
- Algorithm
- Operations
- Software
- Porting to GPU
  - Biogeochemical model (**Fortran**)
  - Model driver (**C**)
- Results
- Remarks

Motivation

- **Global carbon cycle, CO_2 uptake of the world's oceans**
- **Parameter estimation in biogeochemical models**



Simulated concentration of nutrients (phosphate, PO_4^{3-}) at surface layer in $m\,mol\,P/m^3$. The longitudinal and latitudinal resolution is at $1.0°$.

Motivation cont'd

- **Ocean Biogeochemical Dynamics** [Sarmiento and Gruber, 2006]
- System of transport equations for biogeochemical tracers:

$$\frac{\partial y_i}{\partial t} = \underbrace{\nabla \cdot (\kappa \nabla y_i)}_{\text{diffusion}} - \underbrace{\nabla \cdot (v\, y_i)}_{\text{advection}} + \underbrace{q_i(y, \mathbf{u})}_{\text{reaction}}, \quad i = 1, \ldots$$

- **Climatological** (annual periodic) forcing
- Solution is **steady annual periodic state** (equilibrium)
- Involves integration over thousands of model years

Motivation cont'd

- **Transport Matrix Method** [Khatiwala et al., 2005]
- Discretized and approximated problem:

$$\mathbf{y}_{j+1} = \underbrace{\mathbf{A}_{imp,j}\,(\mathbf{A}_{exp,j}}_{\text{transport matrices}}\,\mathbf{y}_j + \Delta t\, q_j(\mathbf{y}_j, \mathbf{u})), \quad j = 0, \ldots$$

- **Monthly** averaged matrices provided
- Interpolation needed:

$$\mathbf{A}_{imp,j} = \alpha_j\, \mathtt{Ai}[i_{\alpha,j}] + \beta_j\, \mathtt{Ai}[i_{\beta,j}] \quad \text{and} \quad \mathbf{A}_{exp,j} = \alpha_j\, \mathtt{Ae}[i_{\alpha,j}] + \beta_j\, \mathtt{Ae}[i_{\beta,j}]$$

Algorithm

- **Assuming:** $1\,\text{year} \equiv 360\,\text{days}$ and $\Delta t \equiv 3\,\text{h}$

1: $\mathbf{y} = \mathbf{y}_0$
2: **repeat**
3: **for** $j = 1, \ldots, 2880$ **do**
4: evaluate biogeochemical model: $\mathbf{y}_q = q_j(\mathbf{y}, \mathbf{u})$
5: interpolate matrices to time step j
6: perform explicit step: $\mathbf{y} = \mathbf{A}_{exp,j}\,\mathbf{y}$
7: perform implicit step: $\mathbf{y} = \mathbf{A}_{imp,j}(\mathbf{y} + \Delta t\,\mathbf{y}_q)$
8: **end for**
9: **until** steady annual cycle is reached

Operations

- Evaluate biogeochemical model:
  - `BGCStep();`
  - Including copying between different data alignments
- Interpolate matrices:
  - `MatCopy();`
  - `MatScale();`
  - `MatAXPY();`
- Apply explicit and implicit step:
  - `MatMult();`
- **99.4 %** of computational effort is spent by these operations.

Porting to GPU

- **Software:**
  - **Metos3D** [Piwonski and Slawig, 2013]
  - `github.com/metos3d`
  - **PETSc** based implementation in **C** [Balay et al., 1997]
  - Using PETSc matrix and vector operations as well as data structures
  - Coupling **biogeochemical models** implemented in **Fortran**
- **Objectives:**
  - Do not change **Fortran** implementation **at all**
  - Do not change **C** implementation, if possible

Biogeochemical model

- Fortran implementation:
  - **PGI CUDA Fortran** compiler license
  - Wrapper file `model.CUF` with Fortran kernels
  - Inclusion of original through: `#include "model.F"`
  - Macro to change `subroutine` to `attributes(device) subroutine`
- Copying between data alignments:
  - **Thrust** (C++) iterators
  - Operator overloading

Model Driver

- **PETSc**-dev
  - GPU enabled PETSc version
  - `MatMult()` is already implemented [Minden et al., 2010]
  - `MatCopy()`, `MatScale()`, `MatAXPY()` had to be added
- **PETSc classes**
- Object oriented library using C language:
  - Data encapsulation
  - Polymorphism
  - Inheritance

PETSc cont'd

- **Class `Mat`**
- Operations:

```
struct _MatOps {
...
PetscErrorCode (*mult)(Mat,Vec,Vec);
PetscErrorCode (*copy)(Mat,Mat,...);
PetscErrorCode (*scale)(Mat,PetscScalar);
PetscErrorCode (*axpy)(Mat,PetscScalar,Mat,...);
...
};
```

PETSc cont'd

- Example: **MatScale** using **CUSP** [Bell and Garland, 2010]
- Own implementation:

```
PetscErrorCode MatScale_SeqAIJCUSP(Mat M, PetscScalar alpha) {
...
cusp::blas::scal(cuspstruct->mat->values, alpha);
...
};
```

- Adding own implementation:

```
M->ops->scale = MatScale_SeqAIJCUSP;
```

Results

- **GPU:** GeForce GTX 480
- **CPU:** Intel Xeon E5520, running at 2.27 GHz, only single core used
- Biogeochemical model: **N-DOP** [Dutkiewicz et al., 2005]

	Min	Max	Avg	StdDev
CPU	621.43 s	626.79 s	622.14 s	0.540
GPU	28.17 s	28.20 s	28.18 s	0.003
	22.06			

Overall performance gain simulating one model year using the N-DOP model at a longitudinal and latitudinal resolution of $2.8125°$.

Results cont'd

- Performance gain per operation:

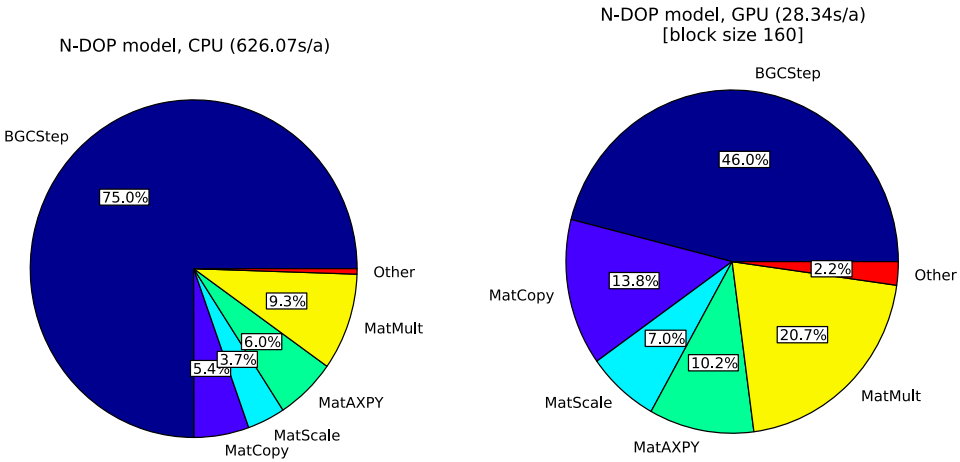
Routine	CPU	GPU	CPU : GPU
`BGCStep`	469.76 s	13.05 s	**36.00**
`MatCopy`	34.04 s	3.91 s	**8.70**
`MatScale`	23.33 s	1.99 s	**11.70**
`MatAXPY`	37.49 s	2.89 s	**12.96**
`MatMult`	58.19 s	5.87 s	**9.92**

- Performance of `MatMult` in detail:
  - **GPU:**

    performance: 11.9 GFlop/s **7 %** of 168 GFlop/s

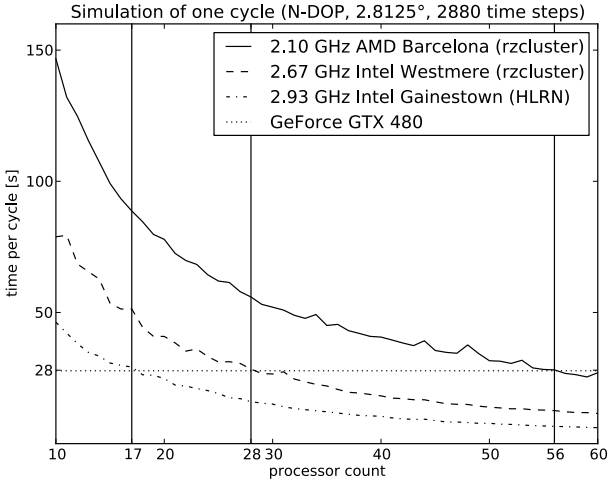
    bandwidth utilization: 119.4 GB/s **67.4 %** of 177 GB/s

Results cont'd



Distribution of computational time per operation during the simulation of one model year.

Results cont'd



Performance of GPU simulation compared to different CPU clusters.

Remarks

- **PGI Fortran preprocessor** didn't like:
  - `#define subroutine attributes(device) subroutine`
  - **C preprocessor** was used
- Only **sequential** PETSc operations were implemented
- Ongoing work on **multi GPU** implementation

Thanks

Thanks for your attention!

Special thanks to Dr. Ulrich Knechtel