Oceanic CO_2-Uptake
Use of PETSc in Climate Research

Piwonski J. , Siewertsen E., Kratzenstein C., Slawig T.

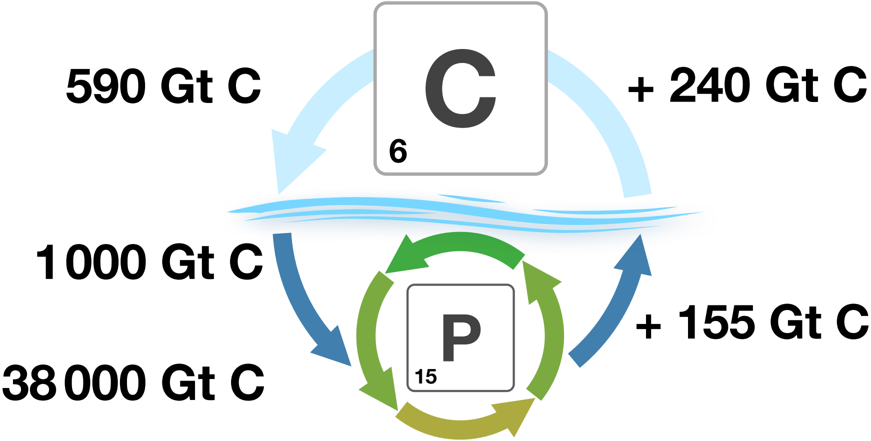
CAU - Christian-Albrechts-Universität zu Kiel

30. Juni 2016

PETSc User Meeting 2016, Vienna

Oceanic CO_2-uptake

[Stocker et al., 2013, IPCC13]



Modelling

- **System of transport equations:**

$$\frac{\partial y_i}{\partial t} = \underbrace{\nabla \cdot (\kappa \nabla y_i)}_{\text{diffusion}} - \underbrace{\nabla \cdot (v\, y_i)}_{\text{advection}} + \underbrace{q_i(y, \mathbf{u}, b, d)}_{\text{bgc model}}, \quad i = 1, \dots, n_y$$

- **Climatological** (annual periodic) forcing

$$\begin{aligned} \kappa(t+1) &= \kappa(t), & b(t+1) &= b(t), & t \in [0, 1[\\ v(t+1) &= v(t), & d(t+1) &= d(t) & \end{aligned}$$

- Solution is a **steady annual cycle** (equilibrium)

$$y(t+1) = y(t)$$

Off-line simulation

- **Transport Matrix Method** [Khatiwala et al., 2005]

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

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

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

Software

[Piwonski and Slawig, 2016]

- **Metos3D**
  - Marine Ecosystem Toolkit for Optimization and Simulation in 3-D
  - `github.com/metos3d`
- **PETSc** based transport driver
- **Programming interface** for biogeochemical models
- **Load balancing**

Algorithm

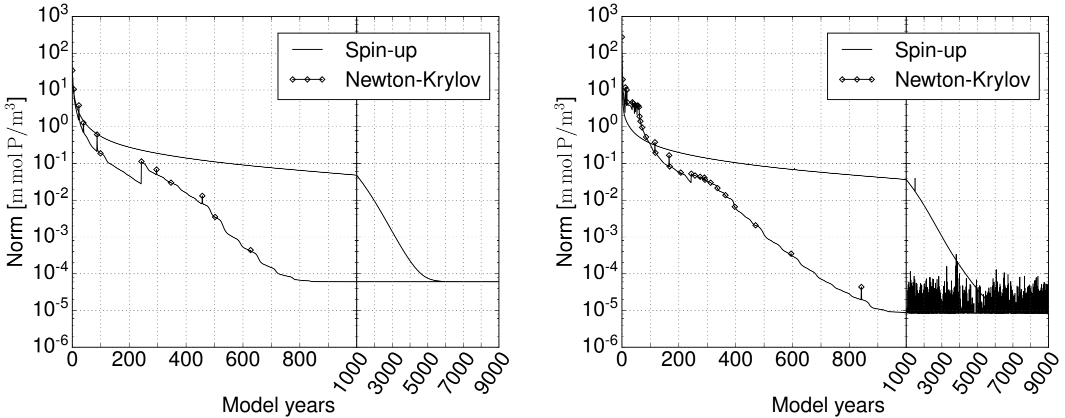
- **Simulation of one model year:**

1: $\mathbf{y} = \mathbf{y}_0$
2: **for** $j = 0, \ldots, n_t - 1$ **do**
3: $\quad \mathbf{q} = \texttt{BGCStep}\,(t_j, \Delta t, \mathbf{y}, \mathbf{u}, \mathbf{b}, \mathbf{d})$
4: $\quad$ interpolate matrices to time step j
5: $\quad$ perform explicit step: $\quad \mathbf{y} = \mathbf{A}_{exp,j}\,\mathbf{y}$
6: $\quad$ perform implicit step: $\quad \mathbf{y} = \mathbf{A}_{imp,j}\,(\mathbf{y} + \mathbf{q})$
7: **end for**

Operations

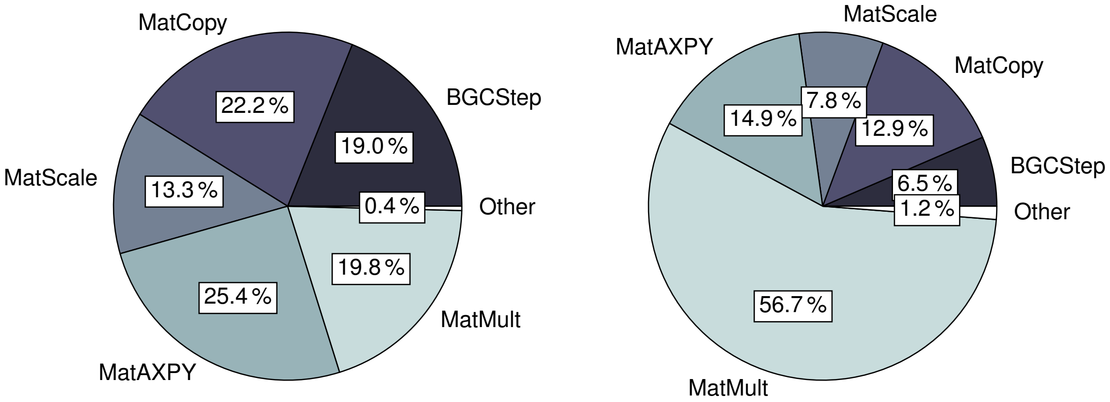
- Evaluate biogeochemical model:
  - **`BGCStep()`**
  - Including copying between different data alignments
- Interpolate matrices:
  - **`MatCopy()`**
  - **`MatScale()`**
  - **`MatAXPY()`**
- Apply matrices:
  - **`MatMult()`**

Solving



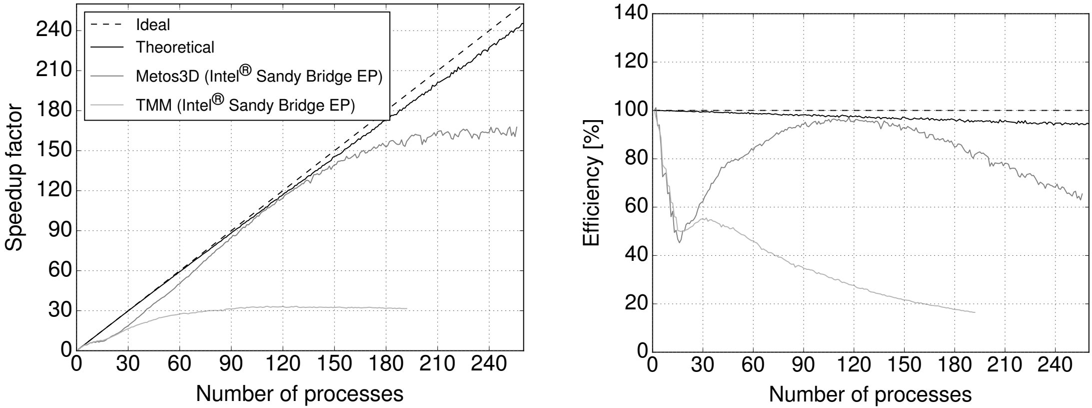
Convergence towards a steady annual cycle using a spin-up and a Newton-Krylov solver.
Left: **N** model. *Right:* **NPZD-DOP** model.

Profilling



Distribution of computational time among main operations.
Left: **N** model. *Right:* **NPZD-DOP** model.

Load balancing



Comparison of theoretical and actual speed-up and efficiency.

Parameter estimation

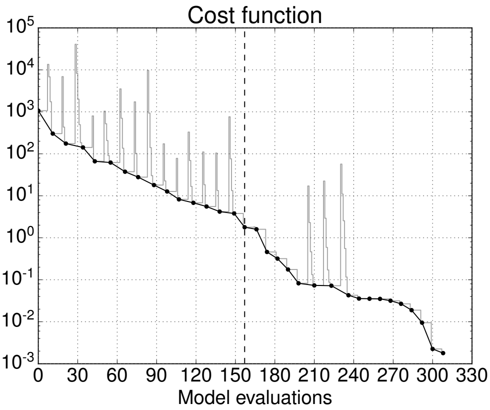
- **Optimization problem:**

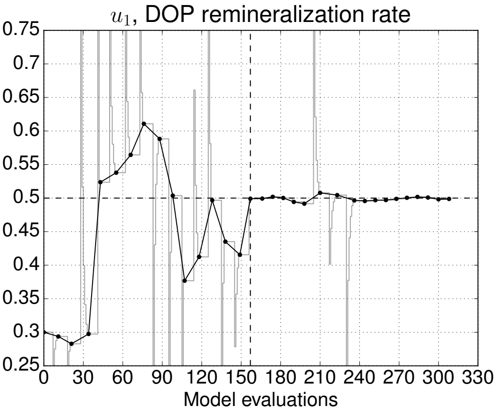
$$\min_{\mathbf{u}\in U} J(\mathbf{u}) \quad s.t. \quad U = \{\mathbf{u} \in \mathbb{R}^m : \mathbf{b}_l \leq \mathbf{u} \leq \mathbf{b}_u\}$$

- **Reduced cost function:**

$$J(\mathbf{u}) = \frac{1}{2}||\mathbf{y}(\mathbf{u}) - \mathbf{y}_d||_2^2$$

Twin experiment





Left: Decay of the (unweighted) cost function. *Right:* Convergence towards the reference parameter.

Migrating to GPUs [Siewertsen et al., 2013]

- **Objectives:**
  - Do not change **Fortran** implementation **at all**
  - Do not change **C** implementation, if possible
- **PETSc**-dev
  - GPU enabled PETSc version
  - **`MatMult()`** is already implemented [Minden et al., 2010]
  - **`MatCopy()`, `MatScale()`, `MatAXPY()`** had to be added

Biogeochemical model

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

Results

- **GPU:** GeForce GTX 480
- **CPU:** Intel Xeon E5520, running at 2.27 GHz, only single core used

	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

- **Optimization problem:**

$$\min_{y,\mathbf{u}} J(y,\mathbf{u}) \quad s.t. \quad G(y,\mathbf{u}) - y = 0$$

- **Cost function:**

$$J(y,\mathbf{u}) = \frac{1}{2}\|y - y_d\|_2^2 + \frac{\alpha}{2}\|\mathbf{u} - \mathbf{u}_g\|_2^2$$

Simultaneous design and optimization [Hamdi and Griewank, 2008]

- **Lagrangian:**

$$L(y, \bar{y}, \mathbf{u}) = J(y, \mathbf{u}) + \bar{y}^\top G(y, \mathbf{u}) - \bar{y}^\top y$$

- **One-shot iteration:**

$$\begin{aligned}
\mathbf{y}_{k+1} &= G(\mathbf{y}_k, \mathbf{u}_k) \\
\bar{\mathbf{y}}_{k+1}^\top &= J_y(\mathbf{y}_k, \mathbf{u}_k) + \bar{\mathbf{y}}_k^\top G_y(\mathbf{y}_k, \mathbf{u}_k) \\
\mathbf{u}_{k+1}^\top &= \mathbf{u}_k - B_k^{-1}(J_u(\mathbf{y}_k, \mathbf{u}_k)^\top + \bar{\mathbf{y}}_k^\top G_u(\mathbf{y}_k, \mathbf{u}_k))
\end{aligned}$$

Automatic differentiation (AD)

- **Adjoint time step:**

$$\bar{\mathbf{y}}_{j-1}^\top = \bar{\mathbf{y}}_j^\top \mathbf{A}_{imp,j}(\mathbf{A}_{exp,j} + \Delta t\, \mathbf{q}_{y,j}(\mathbf{y}_j, \mathbf{u}, \mathbf{b}_j, \mathbf{d}_j)), \quad j = n_t, \dots, 1$$
$$\bar{\mathbf{u}}_{j-1}^\top = \bar{\mathbf{y}}_j^\top \mathbf{A}_{imp,j}\, \Delta t\, \mathbf{q}_{u,j}(\mathbf{y}_j, \mathbf{u}, \mathbf{b}_j, \mathbf{d}_j)$$

- **AD (code transformation) tools for Fortran:**
  - **TAPENADE** [`http://www-tapenade.inria.fr:8080/tapenade/`]
  - **TAF** [`http://www.fastopt.de`]

Code transformation

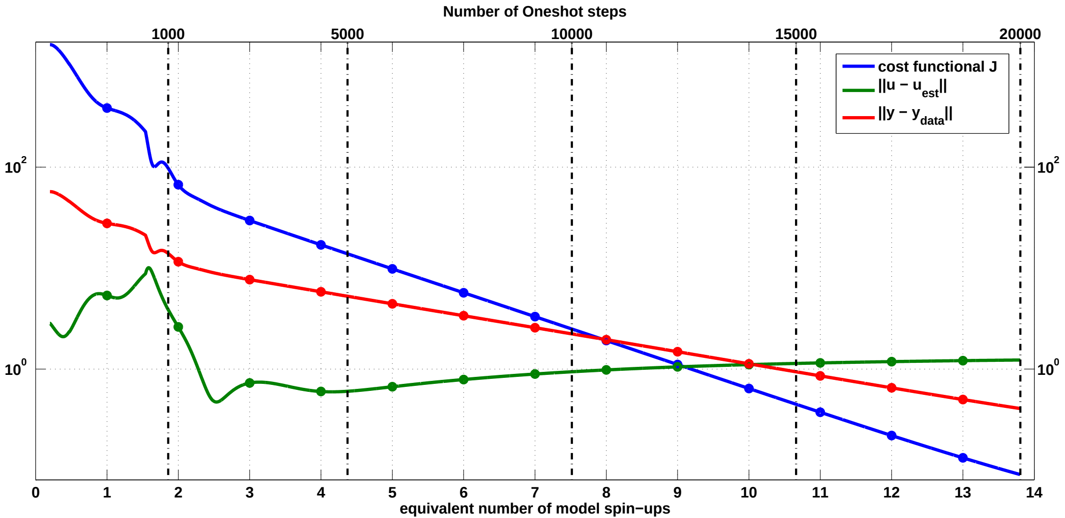
- **Model interface:**

```
subroutine bgc(n, ny, m, nb, nd, dt, q, t, y, u, b, d)
  integer :: n, ny, m, nb, nd
  real*8  :: dt, q(ny, n), t, y(ny, n), u(m), b(nb), d(ny, nd)
end subroutine
```

- **Adjoint model interface:**

```
subroutine bgc_ad(n, ny, m, nb, nd, dt, q, q_ad, t, y, y_ad, u, u_ad, b, d)
  integer :: n, ny, m, nb, nd
  real*8  :: dt, q(ny, n), t, y(ny, n), u(m), b(nb), d(ny, nd)
  real*8  :: q_ad(ny, n), y_ad(ny, n), u_ad(m)
end subroutine
```

Twin experiment



Why we used PETSc ...

- Includes **Fortran**
- Provides **parallel data types**
- Provides **parallelized operations**
- Provides **extended and customizable profilling**
- Provides **robust and flexible Newton-Krylov solver**
- Supports **different platforms** (GPU, co-processors, mobile)