Computer Science in Ocean and Climate Research

Lecture 6: Methods for Parallelization

Prof. Dr. Thomas Slawig

CAU Kiel
Dep. of Computer Science

Summer 2020

Contents

Contents

Modularization in Fortran

- To realize a modular version of the time integration for arbitrary models,
- ... we want to make a call

```
call euler(ydot, tstart, tend, dt, t, y, u)
```

to a time integrator subroutine

```
subroutine euler(f, tstart, tend, dt, t, y, u)
```

where we pass an arbitrary model function (here with two equations):

```
function ydot(y, t, u)
    real(8), intent(in), dimension(:) :: y
    real(8), intent(in) :: t
    real(8), intent(in), dimension(:) :: u
    real(8), dimension(size(y)) :: ydot
    ydot(1) = ...
    ydot(2) = ...
end function
```

Modularization in Fortran

- The model function

  ```
  function ydot(y, t, u)
  ```

  is passed to the time integrator

  ```
  call euler(ydot, tstart, tend, dt, t, y, u)
  ```

- Thus, it has to be declared as input variable (here: `f`) in the signature of the time integrator:

  ```
  subroutine euler(f, tstart, tend, dt, t, y, u)
  ```

- How has this to be done?

Interface blocks

- We need an interface block:

```
subroutine euler(f, tstart, tend, dt, t, y, u)
    interface
        function f(y, t, u)
            real(8), intent(in), dimension(:) :: y
            real(8), intent(in) :: t
            real(8), intent(in), dimension(:) :: u
            real(8), dimension(size(y)) :: f
        end function
    end interface
    real(8), intent(in) :: tstart
    ...
end subroutine euler
```

Interface blocks

- The interface block (left) has to be consistent with the model function to be passed (right):

```
interface
  function f(y, t, u)
    real(8), intent(in), dimension(:) :: y
    real(8), intent(in) :: t
    real(8), intent(in), dimension(:) :: u
    real(8), dimension(size(y)) :: f
  end function
end interface
```

```
function ydot(y, t, u)
   real(8), intent(in), dimension(:) :: y
   real(8), intent(in) :: t
   real(8), intent(in), dimension(:) :: u
   real(8), dimension(size(y)) :: ydot
end function
```

- Usage of `size(y)` as dimension of the return value, both in the model function and the interface, allows model functions with arbitrary dimensions to be passed.
- The dimension of the input `y` determines the one of the function result.
- Now, any model function in conformity with this interface can be integrated with the time integrator.

Contents

Parallelization

- Parallelization on the level of the model and algorithm:
  - in space (domain decomposition)
  - in sub-models: structural (operator splitting)
  - for ensemble runs using sets of parameters, initial values ...
  - in time
- Main question: exchange of data necessary? How often?
  - spatial: usually yes, if the model has spatial processes (e.g., diffusion, transport), in every time-step
  - structural: yes, in every or after a few time step(s)
  - ensemble runs: no
  - in time: yes at the end of time slices

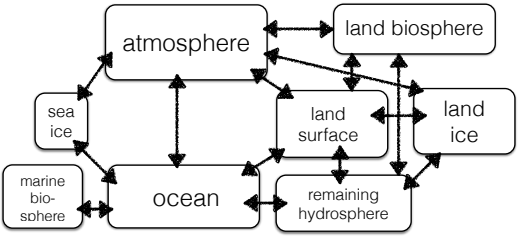
Realization options

- Different computers/processors with no communication:
  $\rightarrow$ ensemble runs
- Different computers/processors with communication over files:
  $\rightarrow$ structural parallelization,
  computing time per model component $\gg$ communication time ($\hat{=}$ amount of data)
- Different processors with separated, **distributed memory**, and internal communication:
  $\rightarrow$ structural and spatial parallelization
  $\rightsquigarrow$ technology (e.g.): MPI (Message Passing Interface)
- Processors with **shared memory**:
  $\rightarrow$ (small scale) ensemble runs
  $\rightarrow$ spatial parallelization with no or small spatial connection
  $\rightsquigarrow$ technology (e.g.): OpenMP (Open Message Passing)
  access on common data possible (has to be taken into account)
- All methods that share or exchange data: results may depend on parallelization scheme.

Contents

Structural parallelization based on operator splitting

- Climate models incorporate different sub-processes/sub-models, ...
- ... realized in different software components (in the ideal case).
- Computational effort may be quite different.
- Example: atmosphere much more expensive than ocean, ...



- ... and radiation most expensive part in atmosphere.
- Moreover: Different algorithms needed for different model parts due to different processes in physics, biology etc. ...
- ... that might also need different spatial and/or temporal resolution,
- vertical and horizontal dimensions quite different in both atmosphere and ocean (ocean: 10 km vs. 40'000 km).

Typical case: spatial processes diffusion and transport/advection

- Recall: Predator-prey model with transport and diffusion, Euler time-stepping:

$$\left.\begin{aligned} x_{k+1} &= x_k + \Delta t\,(Tx_k + Dx_k + x_k * (\alpha - \beta y_k - \lambda x_k)) \\ y_{k+1} &= y_k + \Delta t\,(Ty_k + Dy_k + y_k * (\delta x_k - \gamma - \mu y_k)) \end{aligned}\right\} \quad k = 0, \ldots, n-1,$$

  ... where $x_k * y_k$ means element-wise vector multiplication.

- Processes transport, diffusion, reaction (i.e., predator-prey) are usually treated differently.

⇝ **Operator splitting scheme**:

$$\left.\begin{aligned} x_{k+\frac{1}{2}} &= x_k + \Delta t\,(Tx_k + x_k * (\alpha - \beta y_k - \lambda x_k)) \\ x_{k+1} &= x_{k+\frac{1}{2}} + \Delta t Dx_k \end{aligned}\right\} \text{(analogously for } y\text{)}, k = \ldots$$

- Can be parallelized:

$$\left.\begin{aligned} \text{processor } \#0: \quad \hat{x} &= x_k + \Delta t\,(Tx_k + x_k * (\alpha - \beta y_k - \lambda x_k)) \\ \text{processor } \#1: \quad \bar{x} &= \Delta t Dx_k, \\ x_{k+1} &= \hat{x} + \bar{x}. \end{aligned}\right\} \quad k = \ldots$$

Implicit diffusion

- **Operator splitting scheme**:

$$\left.\begin{aligned} x_{k+\frac{1}{2}} &= x_k + \Delta t\,(Tx_k + x_k * (\alpha - \beta y_k - \lambda x_k)) \\ x_{k+1} &= x_{k+\frac{1}{2}} + \Delta t D x_k \end{aligned}\right\} \text{(analogously for } y\text{)}, k = \ldots$$

- Reason: Diffusion requires small time-step $\Delta t \leq \frac{h^2}{2\kappa}$ (CFL condition, h: spatial step-size).

$\rightsquigarrow$ ... or use implicit method

$$y_{k+1} = y_k + \Delta t f(y_{k+1}, t_{k+1})$$

$\rightsquigarrow$ **Semi-implicit splitting scheme**:

$$\left.\begin{aligned} x_{k+\frac{1}{2}} &= x_k + \Delta t\,(Tx_k + x_k * (\alpha - \beta y_k - \lambda x_k)) \\ x_{k+1} &= x_{k+\frac{1}{2}} + \Delta t D x_{k+1} \Leftrightarrow \text{ solve } (I - \Delta t D)\, x_{k+1} = x_{k+\frac{1}{2}} \end{aligned}\right\} \quad k = \ldots$$

Analogously for y. Now not completely equivalent.

- Needs linear solver in the implicit step.

Structural parallelization for semi-implicit scheme

- Using parallelization for the semi-implicit splitting scheme

$$\left.\begin{array}{l} x_{k+\frac{1}{2}} = x_k + \Delta t\,(Tx_k + x_k * (\alpha - \beta y_k - \lambda x_k)) \\ \text{solve} \quad (I - \Delta t D)\, x_{k+1} = x_{k+\frac{1}{2}} \end{array}\right\} \quad k = \ldots$$

- ... gives:

$$\left.\begin{array}{lll} \text{processor } \#0 & : & \hat{x} = \Delta t\,(Tx_k + x_k * (\alpha - \beta y_k - \lambda x_k)) \\ \text{processor } \#1 & : & \text{solve} \quad (I - \Delta t D)\,\bar{x} = x_k, \\ & & x_{k+1} = \hat{x} + \bar{x}. \end{array}\right\} \quad k = \ldots$$

  Analogously for y.

- Different variables used in implicit step.
- Splitting schemes can be found in every climate model.

Contents

Shared memory parallelization with OpenMP

- Working with directives
- OpenMP directives start with

  ```
  !$omp
  ```

- Encapsule code that shall be parallelized

  ```
  !$omp <omp keyword>
  ...
  !$omp end <omp keyword>
  ```

- Any updates of the changes made to the variables inside are not ensured until the end of the encapsuled work-sharing OpenMP section.
- Optionally define number of threads: `numthreads(4)`
- Default 1, can be set as shell variable, e.g., `OMP_NUM_THREADS=4`.
- More details: `www.openmp.org/wp-content/uploads/F95_OpenMPv1_v2.pdf`

Simple OpenMP examples: do-loop

- Do-loop:

Parallelize all three loops:

```
!$omp do
  do k = 1, 10
    do j = 1, 10
      do i = 1, 10
        A(i,j,k) = ...
      enddo
    enddo
  enddo
!$omp end do
```

Parallelize innermost loop only:

```
do k = 1, 10
  do j = 1, 10
    !$omp do
      do i = 1, 10
        A(i,j,k) = ...
      enddo
    !$omp end do
  enddo
enddo
```

- Branching in and out of the parallelized loop is not allowed.

OpenMP do-loops

- Example: Parallelize using 10 threads:

```
!$omp do
  do k = 1, 1000
     ...
  enddo
!$omp end do
```

- General distribution among teh threads:

  Thread #0: computes $k = 1, \ldots, 100$

  Thread #1: computes $k = 101, \ldots, 200$

  ...

  Thread #9: computes $k = 901, \ldots, 1000$

- However: Distribution of the iterations of the do-loop over the different threads is not predictable.

OpenMP do-loops

- Taking into account the column-wise storing of multi-dimensional arrays in Fortran:

  ```
  real(8) :: A(1:10, 1:10)
  ```

  is stored as

  ```
  (A(1,1), A(2,1),..., A(10,1), A(1,2),...,A(10,2),...,A(10,10))
  ```

- In contrast to: C multi-dimensional arrays are stored row-wise:

  ```
  double  A[10][10];
  ```

  is stored as

  ```
  [A[0][0], A[0][1],..., A[0][9], A[1][0],...,A[1][9],...,A[9][9] ]
  ```

OpenMP do-loops

- Taking into account the storing of multi-dimensional arrays in Fortran results in:

Faster:

```
!$omp do
  do k = 1, 10
    do j = 1, 10
      do i = 1, 10
        A(i,j,k) = ...
      enddo
    enddo
  enddo
!$omp end do
```

Slower:

```
!$omp do
  do i = 1, 10
    do j = 1, 10
      do k = 1, 10
        A(i,j,k) = ...
      enddo
    enddo
  enddo
!$omp end do
```

- ... in general and also using parallelization.

OpenMP sections

- Sections:

```
!$omp sections
!$omp section
  ...
!$omp section
  ...
!$omp end sections
```

- All sections need to be in the same structured block.

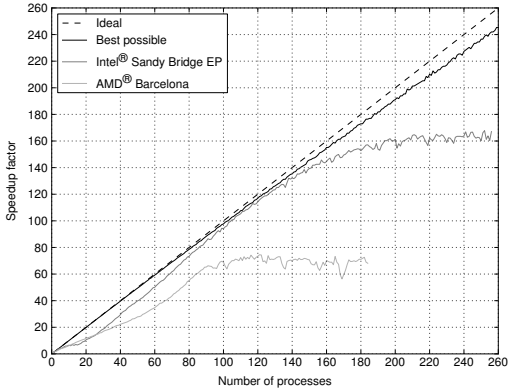
Contents

Parallel-in-time methods

- Idea: Parallelize also time, not only space.
- Spatial parallelization comes to an end, especially for long-time climate simulations:
- Restricted spatial resolution (which is coupled to time-step),
- Example: Marine ecosystem simulation
- Even with efficient load balancing (i.e., distribution of data to processes), more than 128 processes are not effective anymore.
- Difficulty: different length of vertical water profiles in the ocean.

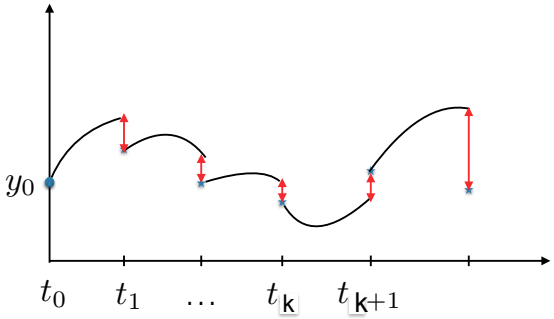
$\rightsquigarrow$ Parallel-in-time methods: current research topic in High Performance Computing.

- Problem: time is sequential.



Idea of the parallel-in-time method(s)

- Split the time interval into sub-intervals (time slices).
- Have two time integrators ...
- ... one "fine", accurate (the original one): F
- ... and a faster, "coarse", inaccurate one: C.
- Compute once with the fast but coarse time integrator ...
- ... to obtain initial values at the beginning of the time slices.



- Compute with the fine solver in parallel on the time slices.
- Perform a correction step ...
- ... and iterate until convergence, i.e., the jumps vanish.

"Pararéel" (parareal) method by Lions Maday Turinici 2001

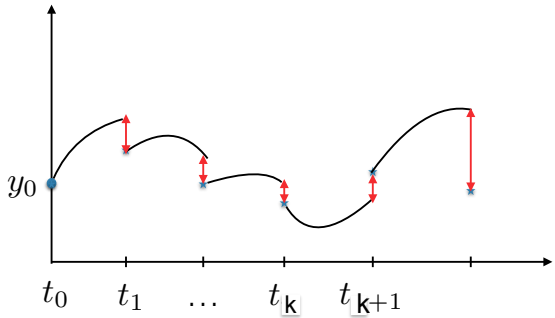
1. $j = 0$: Prediction (coarse) on $[t_0, T]$:

$$y_{k+1}^0 = C(y_k^0), \quad k = 0, 1, \ldots, n-1,$$

$\rightsquigarrow$ gives initial approximations $y_k^0 \approx y(t_k)$.

2. Compute (in parallel)

$$y_{k+1}^c = C(y_k^j), \quad y_{k+1}^f = F(y_k^j).$$



3. Correction step (sequentially):

$$y_{k+1}^{j+1} = C(y_k^{j+1}) + y_{k+1}^f - y_{k+1}^c = C(y_k^{j+1}) + F(y_k^j) - C(y_k^j),$$

4. $j \to j+1$: back to step 2, until a stopping criterion is satisfied.

What is important

- Fortran functions can be passed to other Fortran functions using the explicit interface construct.
- Parallelization is used to accelerate expensive code.
- There are different ways for parallelization.
- Operator splitting schemes are widely used in climate models.
- They can be parallelized using structural parallelization (maybe on top on or additionally to spatial parallelization).
- OpenMP is a technology to parallelize code using shared memory. It can be used, e.g., with Fortran and C.
- Way of application might influence the effect, ...
- ... for example depending on the storage of multi-dimensional arrays.
- Fortran arrays are stored differently than those in C.
- Parallelization in time is another option that is currently being investigated.