A parallel version of the conjugate gradient (CG) method using MPI

- Parallel Programming II -

Manuel Baumann, David Staubach

KTH Stockholm, NADA

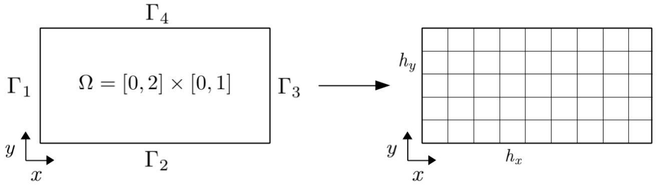
May 22, 2012

Table of Contents

Aim: Solve Poisson's equation

$$-\Delta u = f \quad \text{in } \Omega \subseteq \mathbb{R}^2, \tag{1}$$

where Ω is a two-dimensional, rectangular domain.



The discretization of (1) leads to a linear system of equations

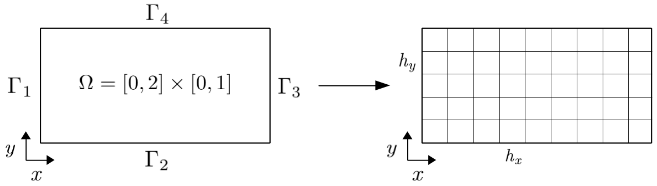
$$A\mathbf{x} = \mathbf{b},$$

with $A \in \mathbb{R}^{N \times N}$ sparse and symmetric positive definite (SPD).

Aim: Solve Poisson's equation

$$-\Delta u = f \quad \text{in } \Omega \subseteq \mathbb{R}^2, \tag{1}$$

where Ω is a two-dimensional, rectangular domain.



The discretization of (1) leads to a linear system of equations

$$A\mathbf{x} = \mathbf{b},$$

with $A \in \mathbb{R}^{N \times N}$ sparse and symmetric positive definite (SPD).

The serial CG algorithm

Use the **conjugate gradient** (CG) method to solve

$$A\mathbf{x} = \mathbf{b},$$

which is an

- iterative,
- Krylov-type,
- memory efficient

solver.

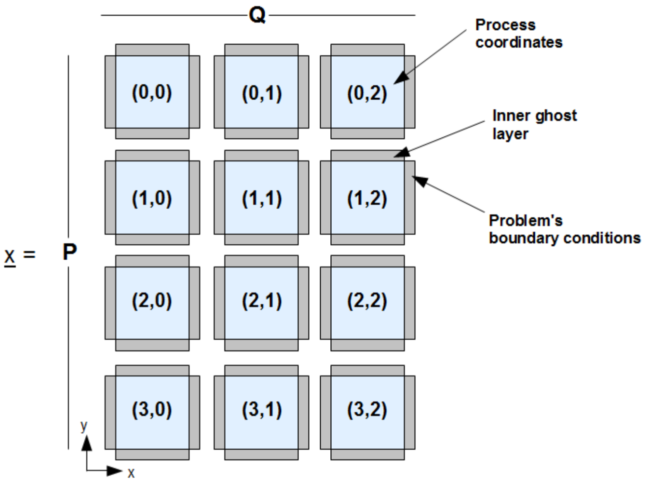
```
Choose initial guess $\mathbf{x}_0$
$\mathbf{r}_0 := \mathbf{b} - A\mathbf{x}_0$
$\mathbf{d}_0 := \mathbf{r}_0$
if $\|\mathbf{r}_0\|_2 < \texttt{tol}$ then
    return
end if
for $k = 0, 1, 2, \ldots, \texttt{MAXITER}$ do
    $\alpha_k := \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{d}_k^T A \mathbf{d}_k}$
    $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{d}_k$
    $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k A \mathbf{d}_k$
    if $\|\mathbf{r}_{k+1}\|_2 < \texttt{tol}$ then
        return
    end if
    $\beta_k := \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$
    $\mathbf{d}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{d}_k$
end for
```

Overview

The parallelization strategy of the CG algorithm

1. pick a problem specific virtual topology
   $\rightarrow$ **process mesh**
2. distribute the vectors' data entries among processes
   $\rightarrow$ **blockwise decomposition**
3. find an efficient communication pattern
   $\rightarrow$ **2D red-black**
4. synchronize the output of the numerical solution
   $\rightarrow$ **global approach**

Introduce a process mesh for each vector's decomposition



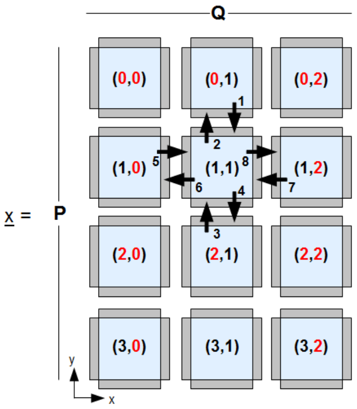
The total number of processes is $R := P \cdot Q$.

How this can be done with MPI

```
MPI_Comm grid_comm;
int dimensions[2];
int wrap_around[2];
int reorder = 1;
dimensions[0] = P;
dimensions[1] = Q;
wrap_around[0] = wrap_around[1] = 0;
MPI_Cart_create( MPI_COMM_WORLD, 2,
dimensions, wrap_around, reorder, &grid_comm );

int grid_rank;
int coords[2];
MPI_Comm_rank( grid_comm, &grid_rank );
MPI_Cart_coords( grid_comm, grid_rank, 2, coords );
```

2D red-black implementation, e.g. process $(1,1)$:



Use MPI built-in function to determine neighbouring processes:

```
MPI_Cart_shift( grid_comm, 0,
1, &N, &S );
MPI_Cart_shift( grid_comm, 1,
1, &W, &E );
```

The communication of

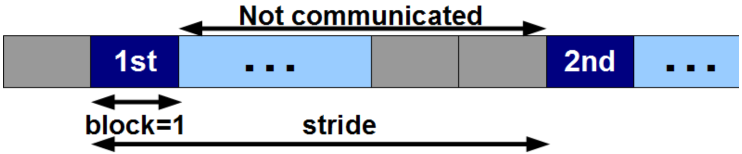
- vector rows is straight forward,
- vector columns is more complex,

because the data values are not successively stored. $\rightarrow$ **Pitfall**

The communication of

- vector rows is straight forward,
- vector columns is more complex,

because the data values are not successively stored. $\rightarrow$ **Pitfall**



MPI helps us to group the column data

```
MPI_Datatype columntype;
MPI_Type_vector( num_elements, block, stride,
MPI_DOUBLE, &columntype );
MPI_Type_commit( &columntype );
```

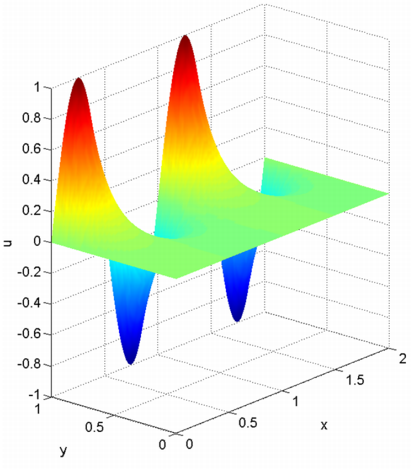
Laplace Equation - Example I

The **correctness of the implementation** can be validated by numerical test calculations.

Solve

$$\begin{aligned} -\Delta u(x,y) &= 0 && \text{in } \Omega \\ u(x,y) &= 0 && \text{on } \Gamma \setminus \Gamma_4 \\ u(x,y) &= \sin(2\pi x) && \text{on } \Gamma_4 \end{aligned}$$

on a 2×2 process mesh with $10,000$ unknowns.

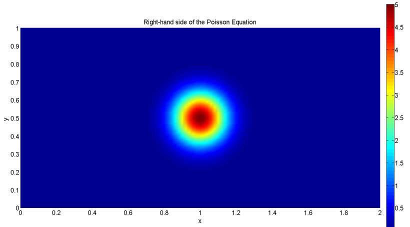


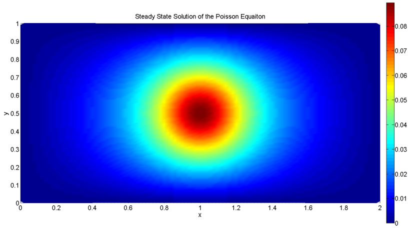
Poisson's Equation - Example II

The **steady state solution** of the heat equation (below) can be obtained by solving

$$\begin{aligned} -\Delta u(x,y) &= f(x,y) \quad && \text{in } \Omega, \\ u(x,y) &= 0 && \text{on } \partial\Omega, \end{aligned}$$

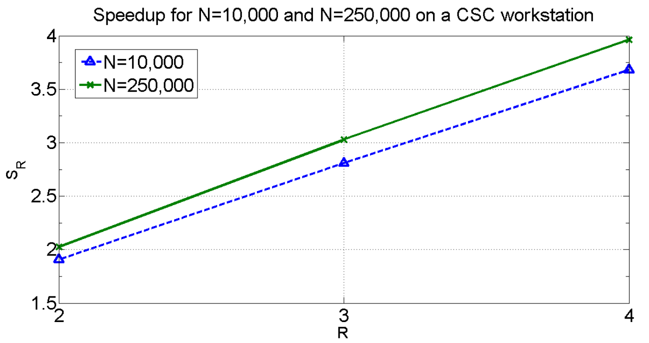
where the source term f is a Gaussian pulse (above).





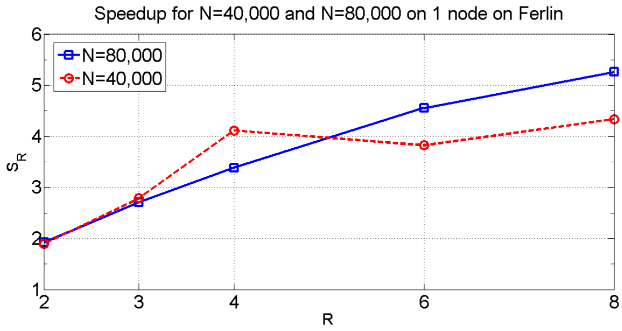
Practical speedup results measured on CSC workstation

Shared-memory multi-core processor with 4 computation cores



- *Ideal* speedup for $N = 250,000$
- Speedup improves for increasing $N \Rightarrow$ theoretical estimation $\checkmark$

Practical speedup results measured on *Ferlin*



- more than 4 processes ⇒ actual communication necessary
- up to 4 processes ⇒ shared-memory possible
- a more detailed knowledge of the hardware architecture is necessary to obtain *ideal* speedup

Conclusion

- Parallelization requires experience with advanced `MPI` routines and techniques
  - Process mesh topology
  - $2D$ red-black communication pattern
  - *column-datatype* for efficient data grouping
- A generic implementation serves as a powerful parallel solver for large *SPD* systems
- Practical speedup investigations
  - *Ideal* results on single multi-core processor, shared-memory machine
  - Hardware-specific optimization necessary for maximum results on cluster machines, e.g. *Ferlin*
- Synchronized output of solution vector requires a rather complex *nested loop-algorithm*

Conclusion

- Parallelization requires experience with advanced MPI routines and techniques
  - Process mesh topology
  - $2D$ red-black communication pattern
  - *column-datatype* for efficient data grouping
- A generic implementation serves as a powerful parallel solver for large *SPD* systems
- Practical speedup investigations
  - *Ideal* results on single multi-core processor, shared-memory machine
  - Hardware-specific optimization necessary for maximum results on cluster machines, e.g. *Ferlin*
- Synchronized output of solution vector requires a rather complex *nested loop-algorithm*

Conclusion

- Parallelization requires experience with advanced `MPI` routines and techniques
  - Process mesh topology
  - $2D$ red-black communication pattern
  - *column-datatype* for efficient data grouping
- A generic implementation serves as a powerful parallel solver for large *SPD* systems
- Practical speedup investigations
  - *Ideal* results on single multi-core processor, shared-memory machine
  - Hardware-specific optimization necessary for maximum results on cluster machines, e.g. *Ferlin*
- Synchronized output of solution vector requires a rather complex *nested loop-algorithm*

Conclusion

- Parallelization requires experience with advanced `MPI` routines and techniques
  - Process mesh topology
  - $2D$ red-black communication pattern
  - *column-datatype* for efficient data grouping
- A generic implementation serves as a powerful parallel solver for large *SPD* systems
- Practical speedup investigations
  - *Ideal* results on single multi-core processor, shared-memory machine
  - Hardware-specific optimization necessary for maximum results on cluster machines, e.g. *Ferlin*
- Synchronized output of solution vector requires a rather complex *nested loop-algorithm*

Thank you for your attention!

Are there any questions or remarks ?