

# Comparative Efficiency Analysis of MPI Blocking and Non-Blocking Communications with Coarray Fortran

**G. Reshetova\* , V. Cheverda, V. Koinov**

Institute of Computational Mathematics and  
Mathematical Geophysics SB RAS,  
Novosibirsk State University

# Content

- Motivation
- Description of approaches
- Numerical implementation
- Conclusion and road map

# Content

- **Motivation**
- Description of approaches
- Numerical implementation
- Conclusion and road map

# Motivation

The **MPI is the most widespread data exchange** interface standard used in parallel programming for clusters and supercomputers with many computer platforms.

The primary means of the MPI communication between processes is passing messages based on basic **point-to-point blocking and non-blocking routines**. The choice of the optimal implementation of exchanges is essential to minimize the idle and transmission times to achieve parallel algorithm efficiency.

We used three realizations of data exchange processes based on blocking, non-blocking point-to-point MPI routines and new features of the **Coarray Fortran** technique to determine the most efficient parallelization strategy.

# Content

- Motivation
- **Description of approaches**
- Numerical implementation
- Conclusion and road map

# Description of approaches. Test problem.

Let us consider the acoustic wave propagation in heterogeneous media, describing a change of the wavefield pressure  $p(x, y, z, t)$  in the domain  $\Omega \times (0, T)$ . Suppose that a wavefield is excited by the source  $f(t)$  located at the point  $(x_s, y_s, z_s)$ . This process in heterogeneous media is defined by the equation

$$\rho \nabla \cdot \left( \frac{1}{\rho} \nabla p \right) - \frac{1}{c^2} \frac{\partial^2 p}{\partial t^2} = -\rho \frac{\partial^2 f}{\partial t^2}, \quad (1)$$

where  $c$  is the velocity of a wave propagating in the medium,  $\rho$  is the density,  $p$  is the acoustic pressure, and  $f$  is the volumetric-type point source.

# Description of approaches. Test problem.

By introducing a displacement velocity vector  $\mathbf{v} = (v_x, v_y, v_z)^T$ , we can represent the second order acoustic wave equation (1) as the first order hyperbolic system

$$\begin{aligned}\rho \frac{\partial \mathbf{v}}{\partial t} + \nabla p &= 0, \\ \frac{1}{\kappa} \frac{\partial p}{\partial t} + \nabla \cdot \mathbf{v} &= \frac{\partial f}{\partial t},\end{aligned}\tag{2}$$

where  $\kappa$  is the adiabatic compression modulus, associated with the velocity by the formula  $v = \sqrt{\kappa/\rho}$ . The resulting system (2) has a symmetric hyperbolic form, and this advantage we use in constructing efficient finite difference schemes on staggered grids.

# Description of approaches.

## Test problem.

By splitting the pressure  $p = (p_x, p_y)^T$  to a two-dimensional case, it is also possible to rewrite (2) as a first order hyperbolic system

$$\begin{aligned}\frac{\partial v_x}{\partial t} &= \frac{1}{\rho} \left( \frac{\partial p_x}{\partial x} + \frac{\partial p_y}{\partial x} \right), \\ \frac{\partial v_y}{\partial t} &= \frac{1}{\rho} \left( \frac{\partial p_x}{\partial y} + \frac{\partial p_y}{\partial y} \right), \\ \frac{\partial P_x}{\partial t} &= -\kappa \frac{\partial v_x}{\partial x} + F_x, \\ \frac{\partial P_y}{\partial t} &= -\kappa \frac{\partial v_y}{\partial y} + F_y.\end{aligned}\tag{3}$$



# Numerical approach

There are several approaches to approximate first order acoustic wave equations (3).

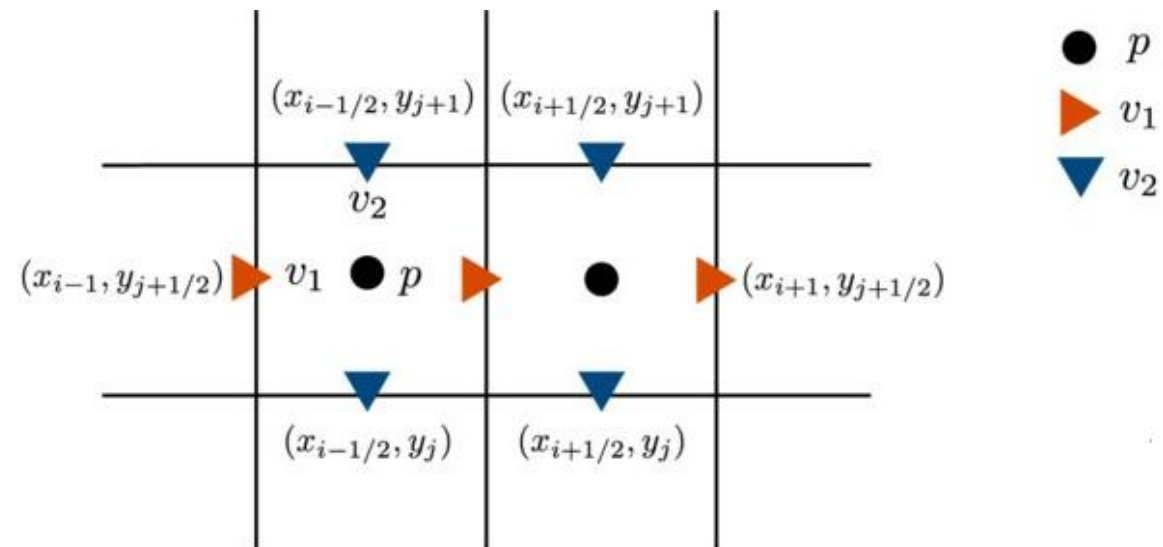
Among the approaches used, we highlight **finite element methods** (FEM), **spectral element methods** (SEM) and the **finite difference method** (FDM).

From a variety of numerical approaches, we choose the finite difference method on staggered grids. The choice of this approach results from the system structure (3): the system equations form a **symmetric first order hyperbolic system** of evolution equations.

In this case, finite difference schemes on staggered grids appear to be the most computationally efficient approach [Virieux1986,Levander1988].

# Finite Difference Method. Staggered grid.

We followed Virieux and developed a second order scheme accurate in space and time on a staggered grid. To excite specific waves, we define a volumetric-type source as product of the Dirac delta function with respect to space



# Finite Difference Method. Staggered grid.

To excite specific waves, we define a volumetric-type source as product of the Dirac delta function with respect to space

$$\delta(x - x_s, y - y_s) \quad (4)$$

and the Ricker wavelet with respect to time

$$f(t) = (1 - 2\pi^2 f_0^2 (t - t_0)^2) \exp[-\pi^2 f_0^2 (t - t_0)^2], \quad (5)$$

where  $f_0$  is the source central frequency and  $t_0$  is the time wavelet delay chosen as  $t_0 = 1/f_0$  s.

# Finite Difference Method. Numerical example.

Figure 1 presents a typical wavefield pressure snapshot excited by the Ricker wavelet impulse.

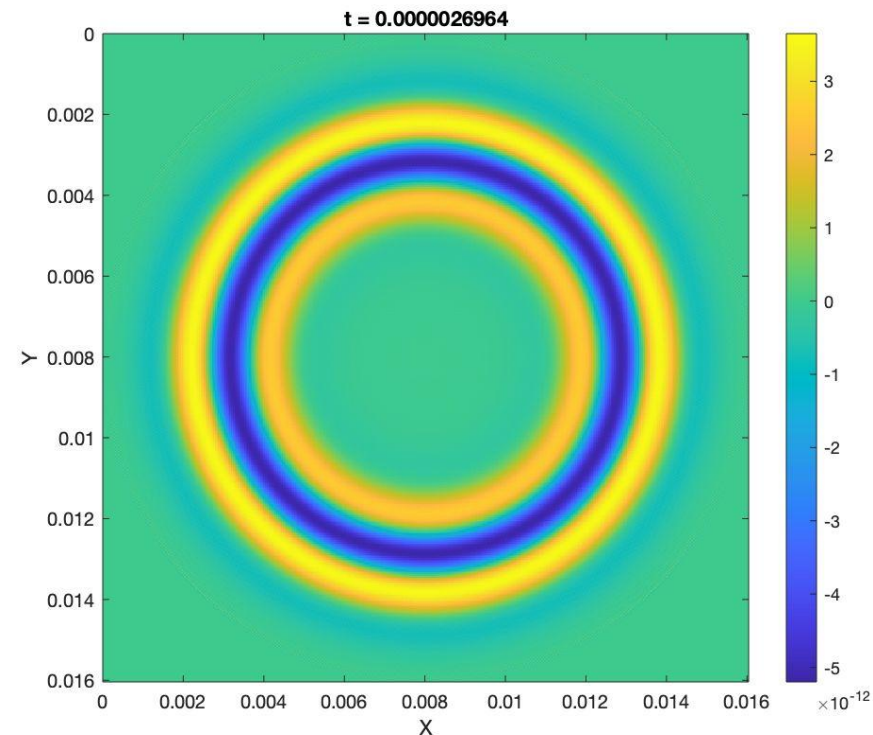
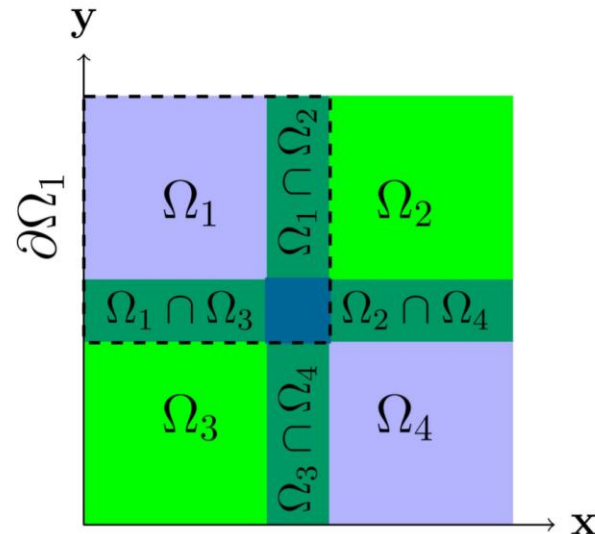


Fig. 1. A snapshot of the pressure component at the time instant  $2.69 \cdot 10^{-6}$ .

# Approach to parallelization. Domain decomposition.

The domain decomposition method is a common approach to parallelization of a computational problem, allowing to split the original problem to several computationally smaller subproblems to be solved independent of each other or related to each other with the help of boundary conditions. This makes the domain decomposition method quite a general and convenient tool for parallel computing.



In turn, the efficiency of a multiprocessor computing system is determined by how evenly the problem solution is distributed over the processes and how much transferring data between processes is minimized.

# Classical MPI Point-to-Point Communications

The elementary MPI routines of data transferring between two processes are the **point-to-point routines** of sending and receiving:

- The **blocking** routines for sending messages between two MPI processes are *MPI\_Send* and *MPI\_Recv*;
- The **non-blocking** transfers *MPI\_Isend* and *MPI\_Irecv*.

The blocking operations are somewhat easier to use. However, the non-blocking can be used like this: call MPI Isend, do some calculations, and then do MPI Wait. This allows computation and communication to overlap, resulting in improved performance overall.

# Point-to-point routines

```
! Data initialization  
! Preparation of the itable () process grid matrix  
! for quick search of related processes see [24]  
do while (t < T)  
MPI_Send(v_x)  
MPI_Recv(v_x)  
MPI_Send(v_y)  
MPI_Recv(v_y)  
! Calculation pressure components p_x, p_y  
MPI_Send(p_x)  
MPI_Recv(p_x)  
MPI_Send(p_y)  
MPI_Recv(p_y)  
! Calculation velocity displacement components v_x, v_y  
end do
```

```
! Data initialization  
! Preparation of the itable () process grid matrix  
! for quick search of related processes see [24]  
do while (t < T)  
MPI_Isend(v_x)  
MPI_Irecv(v_x)  
MPI_Isend(v_y)  
MPI_Irecv(v_y)  
! Calculation pressure components p_x, p_y  
! at the interior points of subdomain  
MPI_Waitall ()  
! Calculation pressure components p_x, p_y  
! at the boundary points  
MPI_Isend(p_x)  
MPI_Irecv(p_x)  
MPI_Isend(p_y)  
MPI_Irecv(p_y)  
! Calculation velocity displacement components v_x, v_y  
! at the interior points of subdomain  
MPI_Waitall ()  
! Calculation velocity displacement components v_x, v_y  
! at the boundary points  
end do
```

# Coarray Fortran

Instead of using the MPI send and receive point-to-point routines, we will take advantage of the new opportunities for exchanging data between processes provided within the **Coarray Fortran approach**.

## History of Coarray Fortran

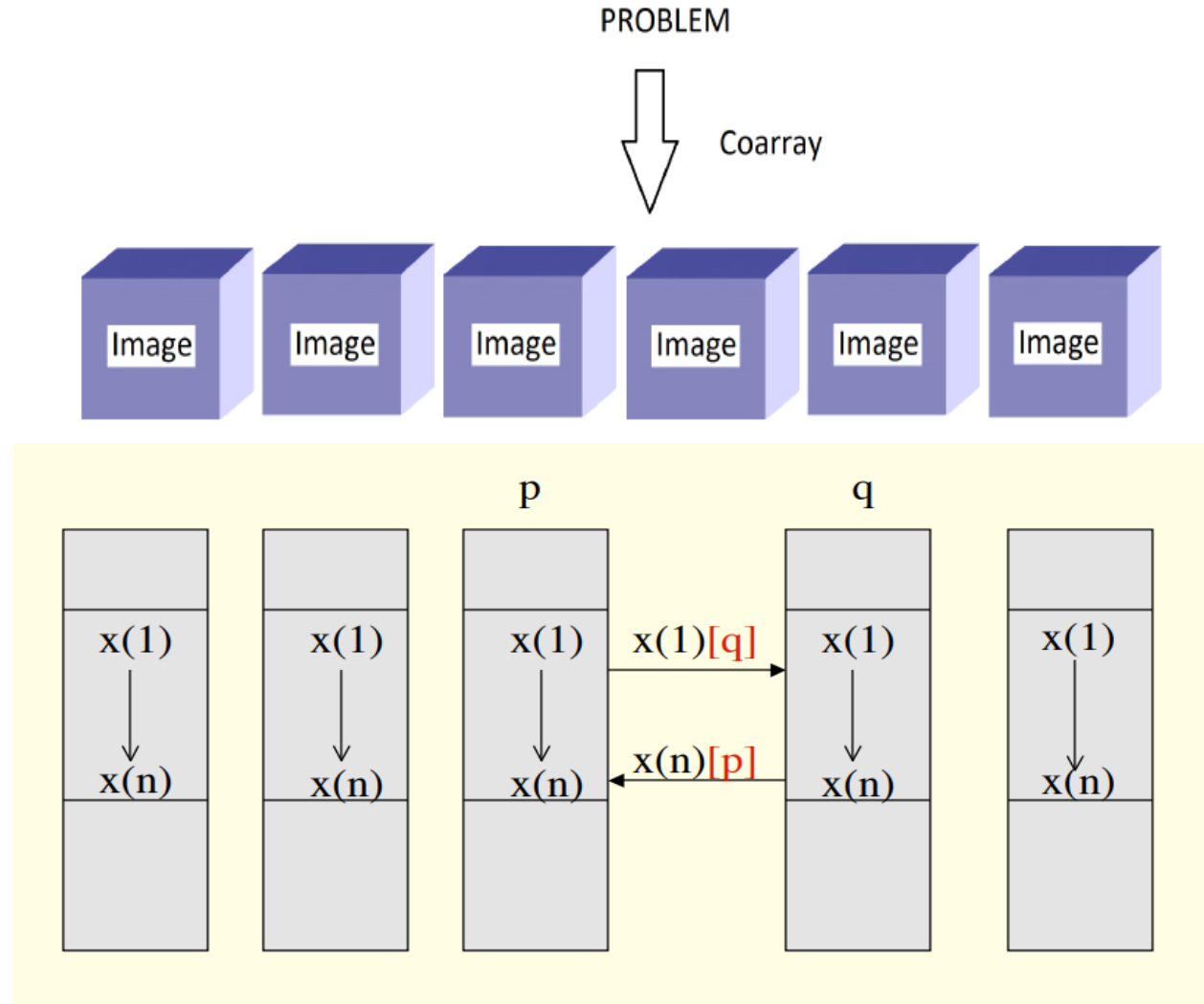
Co-Array Fortran is defined by:

– R.W. Numrich and J.K. Reid, “Co-Array Fortran for Parallel Programming”, ACM Fortran Forum, 17(2):1-31, 1998

- Integrated into Fortran 2008 standard (approved in 2010)
- Additional information on the web: – [www.co-array.org](http://www.co-array.org) – [www.pmodels.org](http://www.pmodels.org)



# Coarray Fortran Memory Model

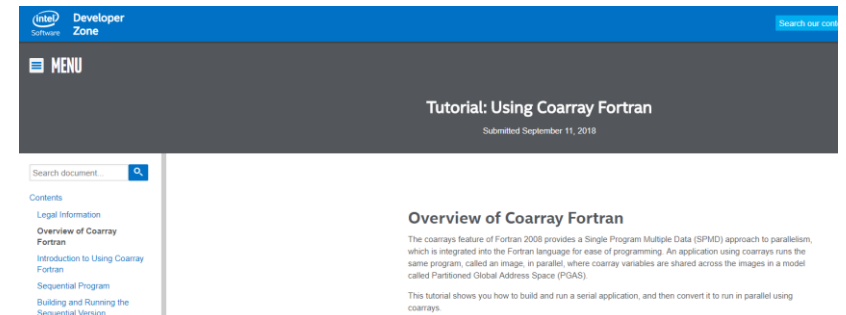


# What is Co-Array Syntax?

Co-Array syntax is a simple extension to normal Fortran syntax.

- It uses normal rounded brackets ( ) to point to data in local memory.
- It uses square brackets [ ] to point to data in remote memory.
- Syntactic and semantic rules apply separately but equally to ( ) and [ ].

```
real :: s[*]  
real :: a(n)[*]  
complex :: z[*]
```



The screenshot shows the Intel Developer Zone website. At the top, there is a blue header with the Intel logo and 'Developer Zone' text. Below the header is a dark grey navigation bar with a 'MENU' button. The main content area is white and features the title 'Tutorial: Using Coarray Fortran' and the date 'Submitted September 11, 2018'. On the left side, there is a 'Search document' input field and a 'Contents' section with a list of links: 'Legal Information', 'Overview of Coarray Fortran', 'Introduction to Using Coarray Fortran', 'Sequential Program', and 'Building and Running the Sequential Version'. The main content area on the right is titled 'Overview of Coarray Fortran' and contains introductory text about the coarrays feature of Fortran 2008, explaining the Single Program Multiple Data (SPMD) approach and the Partitioned Global Address Space (PGAS) model. It also mentions that the tutorial shows how to build and run a serial application and then convert it to run in parallel using coarrays.

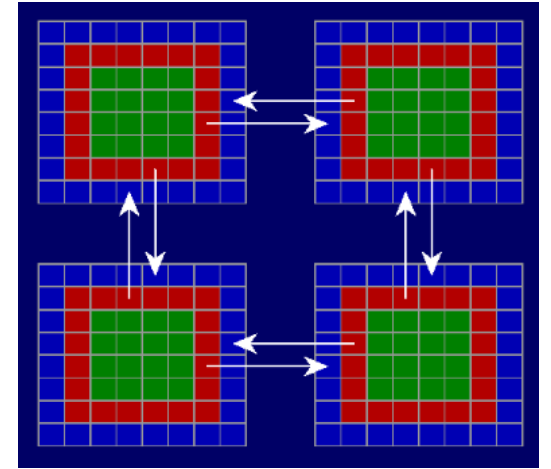
# Declaring and allocating Fortran co-arrays

```
REAL, DIMENSION(:, :)      :: A ! Private array.
REAL, DIMENSION(:, :)[*]  :: B ! Co-array

ALLOCATE ( A( M, N ), STAT = IERR )      ! Allocating private array
ALLOCATE ( B( M, N )[*], STAT = IERR )   ! Allocate co-array.

A(1) = B(2)[7]    ! All images load B(2) from image 7 into their A(1).
A(5) = B(3)      ! All images load their B(3) into their A(5).
```

# Domain Decomposition



MPI	Coarray
<pre> real :: u(0:N+1, 0:M+1) ... call mpi_send(  u(1,1:M), M, mpi_real,  top(myid), tag1,...) call mpi_recv (u(N+1,1:M), M, mpi_real,bottom(myid), tag1,...) call mpi_send(  u(N,1:M), M, mpi_real,bottom(myid), tag2,...) call mpi_recv (  u(0,1:M), M, mpi_real,  top(myid), tag2,...) call mpi_send(  u(1:N,M), N, mpi_real, right(myid), tag3,...) call mpi_recv (  u(1:N,0), N, mpi_real, left(myid), tag3,...) call mpi_send(  u(1:N,1), N, mpi_real, left(myid), tag4,...) call mpi_recv (u(M+1,1:N), N, mpi_real, right(myid), tag4,...) call mpi_waitall(...) </pre>	<pre> real :: u(0:N+1, 0:M+1)[pN,*] ... u(N+1,1)[top(1),top(2)] = u(1,1:M) u(0,1:M)[bottom(1),bottom(2)] = u(N,1:M) u(1:N,0)[right(1),right(2)] = u(1:N,M) u(1:N,M+1)[left(1),left(2)] = u(1:N,1) sync all </pre>

# Domain Decomposition with Coarray

```
! Data initialization  
! Ensuring that data is initialized before  
! being accessed from other images  
sync all  
do while (t < T)  
  if (current_image(1) > 1)  
    v_x(1,:) = v_x(nx-1,:) [current_image(1)-1, current_image(2)]  
  if (current_image(2) > 1) &  
    v_y(:,1) = v_y(:,ny-1) [current_image(1), current_image(2)-1]  
  ! Guarantee that all images are loaded  
  sync all  
  ! Calculation velocity displacement components v_x, v_y  
  if (current_image(1) < number_iprocs)  
    p_x(nx,:) = p_x(2,:) [current_image(1)+1, current_image(2)]  
  if (current_image(2) < number_jprocs)  
    p_y(:,ny) = p_y(:,2) [current_image(1), current_image(2)+1]  
  ! Guarantee that all images are loaded  
  sync all  
  ! Calculation pressure components p_x, p_y  
end do
```

# Testing and Analysis

Testing was carried out on the computers of the Siberian Branch of the Russian Academy of Sciences Siberian Supercomputer Center NKS-1P.

To compare the blocking, non-blocking point-to-point routines and the Coarray Fortran, we carried out the test computations of wavefields propagation through a homogeneous medium.

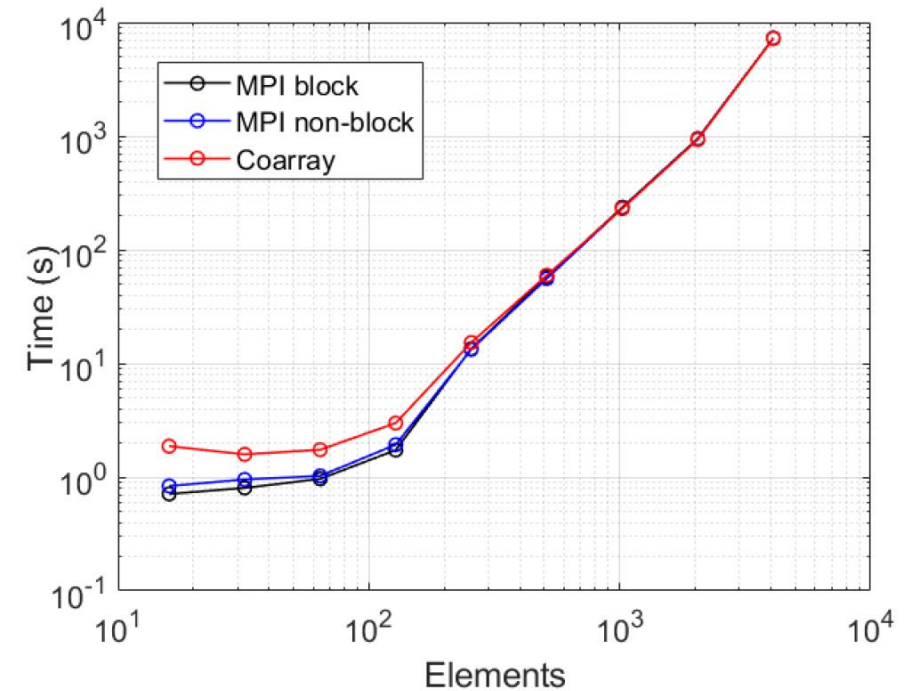
The size of the problem (3) depends on the size of the computational domain. We use the two-dimensional domain decomposition for parallelization and decompose the computational domain into superposition of the equal squares.

We measured the computational times for problems with corresponding to  $16^2, 32^2, 64^2, 128^2, 256^2, 512^2, 1024^2, 2048^2, 4096^2$  elements in each subdomain.

# Testing and Analysis

**Table 1.** The computational time comparison of the blocking, the non-blocking MPI routines and the Coarray Fortran using 32 shared memory processes on one node.

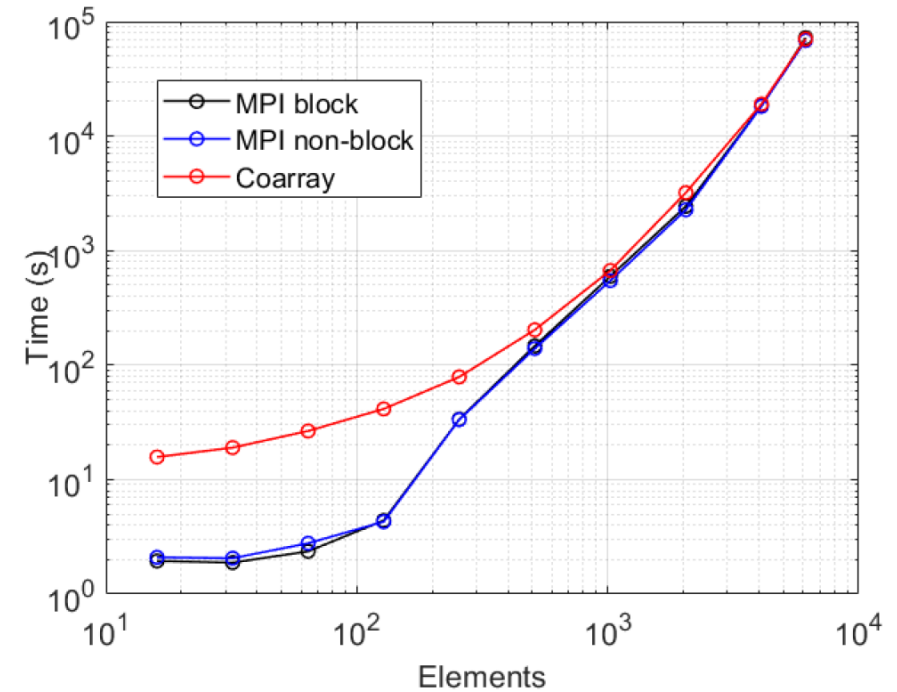
Subdomain elements	Domain elements	Time (s) MPI blocking	Time (s) MPI non-blocking	Time (s) Coarray	Acceleration $K_1$	Acceleration $K_2$
$16^2$	$32 \cdot 16^2$	0.7112	0.8354	1.8702	0.85133	0.38028
$32^2$	$32 \cdot 32^2$	0.8023	0.9521	1.5837	0.84266	0.50660
$64^2$	$32 \cdot 64^2$	0.9644	1.0242	1.7398	0.94161	0.55432
$128^2$	$32 \cdot 128^2$	1.7282	1.9274	2.9875	0.89665	0.57848
$256^2$	$32 \cdot 256^2$	13.4545	13.2655	15.2554	1.01425	0.88195
$512^2$	$32 \cdot 512^2$	58.0052	55.7663	59.4654	1.04015	0.97544
$1024^2$	$32 \cdot 1024^2$	235.748	231.423	229.871	1.01869	1.02557
$2048^2$	$32 \cdot 2048^2$	950.391	938.779	931.316	1.01237	1.02048
$4096^2$	$32 \cdot 4096^2$	7289.87	7224.81	7197.64	1.00901	1.01281



# Testing and Analysis

**Table 2.** The computational time comparison of the blocking, the non-blocking MPI routines and the Coarray Fortran using 81 processes on 3 nodes.

Subdomain elements	Domain elements	Time (s) MPI blocking	Time (s) MPI non-blocking	Time (s) Coarray	Acceleration $K_1$	Acceleration $K_2$
$16^2$	$81 \cdot 16^2$	1.9435	2.0923	15.7352	0.92888	0.12351
$32^2$	$81 \cdot 32^2$	1.8814	2.0553	18.9951	0.91539	0.09905
$64^2$	$81 \cdot 64^2$	2.3584	2.7774	26.5697	0.84914	0.08876
$128^2$	$81 \cdot 128^2$	4.3914	4.2807	41.3869	1.02586	0.10611
$256^2$	$81 \cdot 256^2$	33.6353	33.3587	78.7507	1.00829	0.42711
$512^2$	$81 \cdot 512^2$	147.062	139.319	203.012	1.05558	0.72440
$1024^2$	$81 \cdot 1024^2$	596.154	546.412	669.766	1.09103	0.89009
$2048^2$	$81 \cdot 2048^2$	2438.56	2268.14	3201.47	1.07514	0.76170
$4096^2$	$81 \cdot 4096^2$	18280.8	18191.1	19032.1	1.00493	0.96052
$6144^2$	$81 \cdot 6144^2$	72257.6	68475.1	70315.3	1,05523	1,02762

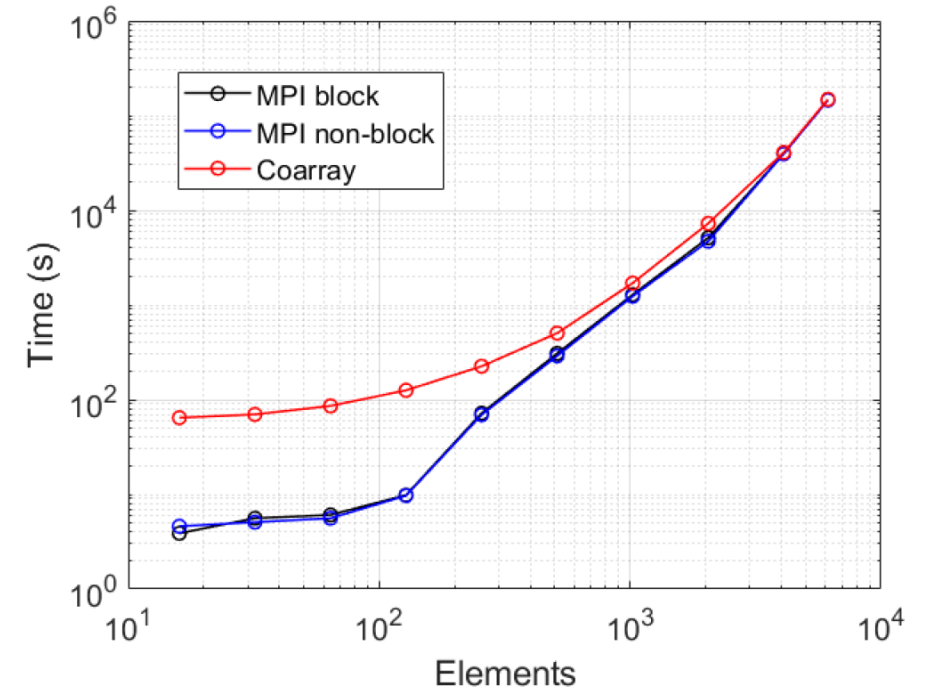




# Testing and Analysis

**Table 3.** The computational time comparison of the blocking, the non-blocking MPI routines and the Coarray Fortran using using 169 processes on 6 nodes.

Subdomain elements	Domain elements	Time (s) MPI blocking	Time (s) MPI non-blocking	Time (s) Coarray	Acceleration $K_1$	Acceleration $K_2$
$16^2$	$169 \cdot 16^2$	3.8883	4.6066	64.4715	0.84407	0.06031
$32^2$	$169 \cdot 32^2$	5.5972	5.0899	69.8915	1.09967	0.08008
$64^2$	$169 \cdot 64^2$	6.0715	5.5972	85.9082	1.08474	0.07067
$128^2$	$169 \cdot 128^2$	9.8369	9.6963	125.546	1.01450	0.07835
$256^2$	$169 \cdot 256^2$	71.9304	69.1176	225.179	1.04070	0.31944
$512^2$	$169 \cdot 512^2$	308.099	289.368	503.697	1.06473	0.61168
$1024^2$	$169 \cdot 1024^2$	1286.59	1235.63	1701.49	1.04124	0.75615
$2048^2$	$169 \cdot 2048^2$	5112.81	4698.08	7261.31	1.08828	0.70412
$4096^2$	$169 \cdot 4096^2$	39476.1	39346.7	40761.5	1.00329	0.96847
$6144^2$	$169 \cdot 6144^2$	146576.0	145210	148365.0	1,00941	0,987941



# Conclusion and road map

We have compared various data exchanges between processes based on the **blocking**, **the nonblocking MPI** point-to-point routines, as well as the new features of the Fortran language based on the **Coarray** Fortran technique.

For the study, a **two-dimensional wave equation** that describes the propagation of acoustic waves in inhomogeneous media and written down in the form of a first order hyperbolic system for the vector of displacement velocity and pressure was taken as a test problem. As a numerical solution method, the method of **finite differences** of the second order of accuracy in spatial variables and time on staggered grids was used to numerically solve the problem.

The **domain decomposition** method was used for parallelization into processes to divide the computational domain into smaller subdomains with overlapping.

We change the problem sizes as well as **various communication approaches** to exchange data between processes. For each version, we have measured the computation time and the acceleration factor.

# Conclusion and road map

We have revealed **the advantages of the delayed nonblocking MPI Isend, MPI Irecv and the Coarray Fortran** when the problem size (the number of elements) increases. For the delayed nonblocking routines, the efficiency can be explained by overlapping computations against the data transfer background. In the Coarray Fortran, the speedup is achieved because the shared memory Coarray variables are read and written from any image, so there is no need to organize data transfers. The graphs show that this approach will be preferable to all others with an increase in the problem dimension.

# Acknowledgements

The research have been supported by the Russian Science Foundation, project 20-11-20112.

The research is carried out using the equipment of the shared research facilities of HPC computing resources at the Joint Supercomputer Center of RAS and the Siberian Supercomputer Center.

Thank you for attention. Questions ?