Multigrid Solver for GBS

Trach-Minh Tran, Federico Halpern
CRPP/EPFL

v1.0, June 2015

Contents

1 The PDE

The PDE considered is

$$\left[\frac{\partial^2}{\partial x^2} + \tau\frac{\partial^2}{\partial x\partial y} + \frac{\partial^2}{\partial y^2} - a(x,y)\right] u(x,y) = f(x,y), \qquad 0 \le x \le L_x,\ 0 \le y \le L_y. \tag{1}$$

On the four boundaries, homogeneous Dirichlet boundary condition $u = 0$ as well as Neumann boundary condition $\partial u/\partial n = 0$ can be applied.

2 Discretization

The grid points (x_i, y_j) are defined by

$$\begin{aligned} x_i &= ih_x = i\frac{L_x}{N_x}, \quad i = 0,\dots,N_x \\ y_j &= jh_y = j\frac{L_y}{N_y}, \quad j = 0,\dots,N_y \end{aligned} \tag{2}$$

Second order Finite Difference discretization of Eq.1 leads to the following 9-point stencil

$$S_{ij} = \frac{1}{h_x^2}\begin{bmatrix} -\tau\alpha/4 & \alpha^2 & \tau\alpha/4 \\ 1 & -2(1+\alpha^2) - h_x^2 a_{ij} & 1 \\ \tau\alpha/4 & \alpha^2 & -\tau\alpha/4 \end{bmatrix}, \qquad \text{where } \alpha = h_x/h_y. \tag{3}$$

Note that the mesh aspect ratio α results in the same stencil for the *anisotropic* Poisson equation with $h_x = h_y$:

$$\frac{\partial^2 u}{\partial x^2} + \alpha^2\frac{\partial^2 u}{\partial y^2} = f. \tag{4}$$

It is shown in [1, p. 119] that this anisotropy can degrade the performance of multigrid using standard relaxations such as Gauss-Seidel or damped Jacobi can be strongly degraded.

With the *lexicographic* numbering

$$I = j(N_x + 1) + i + 1, \tag{5}$$

for the $(N_x+1)(N_y+1)$ nodes, the discretized problem can be expressed as a matrix problem

$$\mathbf{A}\mathbf{u} = \mathbf{f}, \tag{6}$$

where $\mathbf{A}$ is a 9-diagonal matrix, assembled using the stencil defined above. Homogeneous Dirichlet boundary condition can be imposed, for example, on the face $j = 0$ simply by *clearing* the matrix rows and columns $1, 2, \ldots, N_x + 1$, and setting the diagonal terms to 1.

Neumann boundary condition $\partial u/\partial x = 0$ at the face $i = 0$, can be simply implemented by imposing $u_{-1j} = u_{1j}$. The stencil for the boundary nodes $(0, j)$ can thus be modified as

$$S_{0j} = \frac{1}{h_x^2}\begin{bmatrix} 0 & \alpha^2 & 0 \\ 0 & -2(1+\alpha^2) - h_x^2 a_{0j} & 2 \\ 0 & \alpha^2 & 0 \end{bmatrix}. \tag{7}$$

Two model problems are considered in this report:

`DDDD` **problem:** Homogeneous Dirichlet BC at all the 4 boundaries. The *analytic solution* is

$$u(x,y) = \sin\frac{2\pi k_x x}{L_x}\sin\frac{2\pi k_y y}{L_y}, \qquad \text{where } k_x,\ k_y \text{ are positive integers.} \tag{8}$$

`NNDD` **problem:** Neumann boundary conditions $\partial u/\partial x = 0$ at $x = 0$ and $x = L_x$, homogeneous Dirichlet BC at $y = 0$ and $y = L_y$. The *analytic solution* is

$$u(x,y) = \cos\frac{2\pi k_x x}{L_x}\sin\frac{2\pi k_y y}{L_y}, \qquad \text{where } k_x,\ k_y \text{ are positive integers.} \tag{9}$$

In both problems, a depends only on x:

$$a(x,y) = \exp\left[-\frac{(x - L_x/3)^2}{(L_x/2)^2}\right]. \tag{10}$$

The sparse direct solver MUMPS [2] is used to solve (6) in order to check the convergence of the schema described above. Fig.1 shows the expected *quadratic* convergence of the error with respect to h_x with fixed $\alpha = h_x/h_y = 0.5$ for both problems, when the grid size is varied from 16×64 to 512×2048.

3 Multigrid V-cycle

Given an approximate $\mathbf{u}^h$ and right hand side $\mathbf{f}^h$ defined at some grid level represented by the grid spacing h, the following MG V-cycle procedure

$$\boxed{\mathbf{u}^h \leftarrow MG^h(\mathbf{u}^h, \mathbf{f}^h)}$$

will compute a *new* $\mathbf{u}^h$. It is defined recursively by the following steps:

1. If h is the coarsest mesh size,
   - Direct solve $\mathbf{A}^h\mathbf{u}^h = \mathbf{f}^h$
   - Goto 3.
2. Else
   - Relax $\mathbf{u}^h$ ν_1 times.

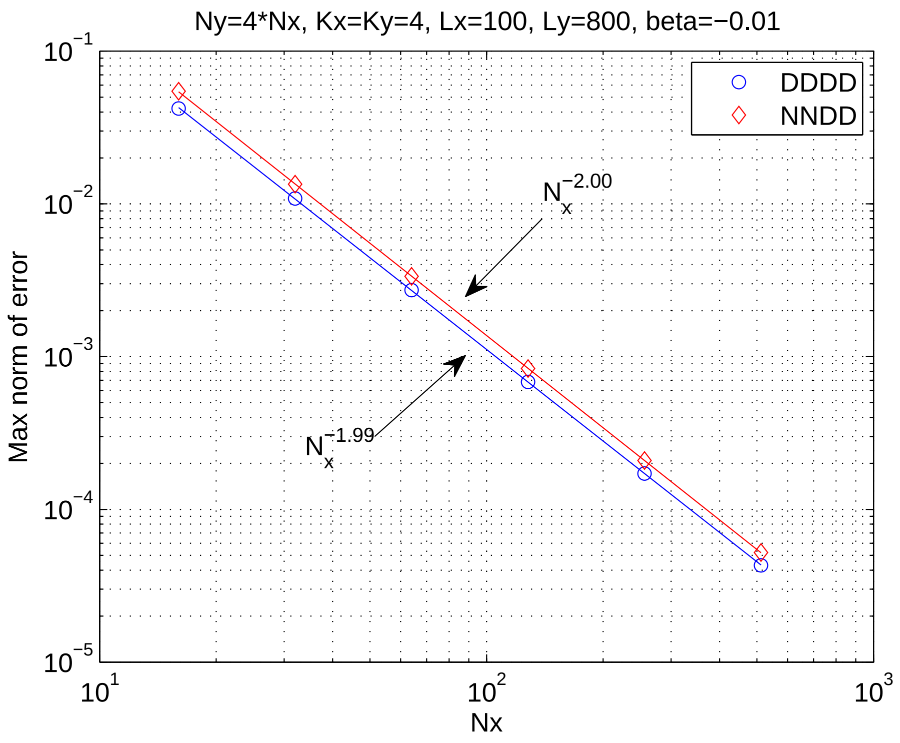


Figure 1: Convergence of the error $\|u_{calc} - u_{anal}\|_\infty$ wrt the number of intervals in the x direction N_x for $L_x = 100$, $L_y = 800$, $k_x = k_y = 4$, $\tau = 1$ and $N_y = 4N_x$.

- $\mathbf{f}^{2h} \leftarrow \mathbf{R}(\mathbf{f}^h - \mathbf{A}^h\mathbf{u}^h)$.
- $\mathbf{u}^{2h} \leftarrow MG^{2h}(\mathbf{u}^{2h}, \mathbf{f}^{2h})$ μ times.
- $\mathbf{u}^h \leftarrow \mathbf{u}^h + \mathbf{P}\mathbf{u}^{2h}$.
- Relax $\mathbf{u}^h$ ν_2 times.

3. Return

In the procedure above, the operators $\mathbf{R}$ and $\mathbf{P}$ denote respectively the *restriction* (from *fine* to *coarse* grid) and the *prolongation* (from *coarse* to *fine* grid). Notice that in this multigrid procedure, $\mathbf{R}$ applies only to the *right hand side* while $\mathbf{P}$ applies only to the *solution*. The standard $V(\nu_1, \nu_2)$ cycle is obtained by calling this MG^h procedure with $\mathbf{f}^h$ defined at the *finest* grid level, a guess $\mathbf{u}^h = 0$ and $\mu = 1$, while $\mu = 2$ results in the $W(\nu_1, \nu_2)$ cycle.

Details on the grid coarsening, the inter-grid transfers and methods of relaxation are given in the following.

3.1 Grid coarsening

Let start with the one-dimensional *fine* grid defined by x_i, $i = 0, \ldots, N$, assuming that N is even. The next coarse grid (with $N/2$ intervals) is obtained by simply discarding the grid points with *odd* indices.

In order to get a *smallest coarsest* grid (so that it is possible to solve *cheaply* the problem with a *direct* method), N should be $N = N_c 2^{L-1}$ where L the total number of grid levels and N_c is either 2 or a *small odd* integer. As an example, the fine grid with $N = 768$ can have up to 9 grid levels, and a coarsest grid with 3 intervals, see Table 1.

For a two-dimensional grid, the same procedure is applied to both dimensions. The result of such procedure is illustrated in Fig. 2, for a 8×4 fine grid.

L	N				
1	2	3	5	7	9
2	4	6	10	14	18
3	8	12	20	28	36
4	16	24	40	56	72
5	32	48	80	112	144
6	64	96	160	224	288
7	128	192	320	448	576
8	256	384	640	896	1152
9	512	768	1280	1792	2304
10	1024	1536	2560	3584	4608

Table 1: Set of values of the *fine* grid number of intervals N to obtain a *coarsest* grid size at most equal to 9 with at most 10 grid levels.

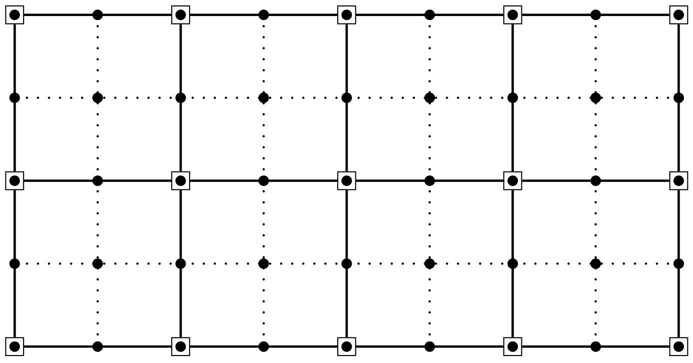


Figure 2: A *coarse* 4×2 grid (□) obtained from a 8×4 fine grid (●).

3.2 Inter-grid transfers

The one-dimensional *prolongation* operator for the second-order FD discretization is chosen the same as the one obtained with the *linear Finite Elements* [3]. For a $N = 8$ grid, it can be represented as a 9×5 matrix given by

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{11}$$

The *restriction* matrix $\mathbf{R}$ is simply related to $\mathbf{P}$ by

$$\mathbf{R} = \frac{1}{2}\mathbf{P}^T = \frac{1}{2}\begin{pmatrix} 1 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 1 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1 \end{pmatrix}. \tag{12}$$

For Dirichlet BC imposed on the *left* boundary one has to set $P_{21} = R_{12} = 0$, while for Dirichlet BC imposed on the *right* boundary, $P_{N,N/2+1} = R_{N/2+1,N} = 0$. Notice that these inter-grid operators are identical to the standard *linear interpolation* and *full weighting* operators.

For a two-dimensional problem, using the property that the grid is a *tensor product* of two one-dimensional grids, the restriction of the right hand side f^h_{ij} and the prolongation of the solution u^{2h}_{ij} can be computed as

$$\begin{aligned} \mathbf{f}^{2h} &= \mathbf{R}^x \cdot \mathbf{f}^h \cdot (\mathbf{R}^y)^T \\ \mathbf{u}^h &= \mathbf{P}^x \cdot \mathbf{u}^{2h} \cdot (\mathbf{P}^y)^T \end{aligned} \tag{13}$$

3.3 Relaxations

Gauss-Seidel and damped Jacobi iterations are used as relaxation methods in the multigrid V cycle. In general, Gauss-Seidel method is more efficient but much more difficult to *parallelize* than the Jacobi method.

It should be noted that if $a(x, y)$ in Eq. 1 is non-positive, both relaxations diverge! This can be seen by considering the following one-dimensional FD equation with uniform a:

$$u_{j-1} - (2 + ah^2)u_j + u_{j+1} = h^2 f_j. \tag{14}$$

Using the damped Jacobi relaxation, the error $\epsilon^{(m)}_j \equiv u_{anal}(x_j) - u^{(m)}_j$ at iteration $m + 1$ is given by

$$\epsilon^{(m+1)}_j = \frac{\omega}{2 + h^2 a}(\epsilon^{(m)}_{j-1} + \epsilon^{(m)}_{j+1}) + (1 - \omega)\epsilon^{(m)}_j. \tag{15}$$

Performing a *local mode analysis* (or Fourier analysis) (see [1, p. 48]), assuming that $\epsilon^{(m)}_j = A(m)e^{ij\theta}$, where θ is related to the mode number k by $\theta = 2\pi k/N$, the *amplification factor* $G(\theta)$ is obtained as

$$\begin{aligned} G(\theta) &= \frac{A(m+1)}{A(m)} = \frac{2\omega}{2 + h^2 a}\cos\theta + (1 - \omega) \\ &= G_0(\theta) - \frac{\omega h^2 a}{2 + h^2 a}\cos\theta \simeq G_0(\theta) - \frac{\omega h^2 a}{2}\cos\theta, \\ G_0(\theta) &= 1 - 2\omega \sin^2\frac{\theta}{2}, \end{aligned} \tag{16}$$

where $G_0(\theta)$ is the amplification factor for $a = 0$. Note that $|G_0(\theta)| < 1$ for *all* θ and $0 < \omega < 1$ but $\max_{|\theta|<\pi} |G(\theta)| > 1$ if $a < 0$.

In Gauss-Seidel relaxation method, the errors evolve as:

$$\epsilon^{(m+1)}_j = \frac{\epsilon^{(m+1)}_{j-1} + \epsilon^{(m)}_{j+1}}{2 + h^2 a}. \tag{17}$$

Applying again the same Fourier analysis yields the following complex amplification factor:

$$\begin{aligned} G(\theta) &\simeq G_0(\theta)\left(1 - \frac{h^2 a}{2 - e^{-i\theta}}\right) \\ G_0(\theta) &= \frac{e^{i\theta}}{2 - e^{-i\theta}}, \quad |G_0(\theta)| < 1 \end{aligned} \tag{18}$$

which show that the Gauss-Seidel relaxations *diverge* if $a < 0$.

Notice that when $a > 0$, the effect of a on the amplification is negligible and is thus ignored in the following two-dimensional analysis. Applying the damped Jacobi scheme on the stencil (3), the error at the iteration $m + 1$ is given by:

$$\begin{aligned} \epsilon^{(m+1)}_{ij} = &\frac{\omega}{2(1+\alpha^2)}\left[\epsilon^{(m)}_{i-1,j} + \epsilon^{(m)}_{i+1,j} + \alpha^2(\epsilon^{(m)}_{i,j-1} + \epsilon^{(m)}_{i,j+1}) + \frac{\tau\alpha}{4}(\epsilon^{(m)}_{i+1,j+1} + \epsilon^{(m)}_{i-1,j-1} - \epsilon^{(m)}_{i-1,j+1} - \epsilon^{(m)}_{i+1,j-1})\right] \\ &+ (1 - \omega)\epsilon^{(m)}_{ij}. \end{aligned} \tag{19}$$

Using the two-dimensional Fourier mode expression

$$\epsilon_{ij}^{(m)} = A(m)e^{i(\theta_1+\theta_2)}, \quad -\pi < \theta_1, \theta_2 \leq \pi, \tag{20}$$

the amplification factor $G = A(m+1)/A(m)$ is given by

$$G(\theta_1, \theta_2; \omega, \alpha, \tau) = 1 - \frac{2\omega}{1+\alpha^2}\left(\sin^2\frac{\theta_1}{2} + \alpha^2 \sin^2\frac{\theta_2}{2} + \frac{\tau\alpha}{4}\sin\theta_1\,\sin\theta_2\right). \tag{21}$$

The errors in Gauss-Seidel method, assuming a *lexicographic* ordering for the unknowns (increasing first i then j), are updated according to

$$\epsilon_{ij}^{(m+1)} = \frac{1}{2(1+\alpha^2)}\left[\epsilon_{i-1,j}^{(m+1)} + \epsilon_{i+1,j}^{(m)} + \alpha^2(\epsilon_{i,j-1}^{(m+1)} + \epsilon_{i,j+1}^{(m)}) + \frac{\tau\alpha}{4}(\epsilon_{i+1,j+1}^{(m)} + \epsilon_{i-1,j-1}^{(m+1)} - \epsilon_{i-1,j+1}^{(m)} - \epsilon_{i+1,j-1}^{(m+1)})\right]. \tag{22}$$

The Fourier mode analysis then leads to the following complex amplification factor

$$G(\theta_1, \theta_2; \alpha, \tau) = \frac{e^{i\theta_1} + \left(\alpha^2 + i\frac{\tau\alpha}{2}\sin\theta_1\right)e^{i\theta_2}}{2(1+\alpha^2) - \left[e^{-i\theta_1} + \left(\alpha^2 - i\frac{\tau\alpha}{2}\sin\theta_1\right)e^{-i\theta_2}\right]}. \tag{23}$$

Curves of G for *fixed* θ_2 are plotted in Fig. 3 showing *convergence* ($\max|G| < 1$) for $\tau = -1, 0, 1, 2$, using the damped Jacobi method. The same conclusions are obtained for Gauss-Seidel relaxations as shown in Fig. 4 where the absolute values of the complex amplification factor G are plotted. However, for larger $|\tau| > 2$, both methods diverge as can be seen in Fig. 5.Notice however that the PDE (1) is *elliptic* only when $|\tau| < 2$ is satisfied!

In summary, the local mode analysis predicts that

- Negative values of the coefficient a and large mixed derivative ($|\tau| > 2$) can make both damped Jacobi and Gauss-Seidel relaxations diverge.
- Positive values of a can decrease the amplification factor (improving thus the convergence rate) but its contributions $h^2 a$ decrease for increasing grid resolution.

These predictions will be checked against numerical experiments in the next section.

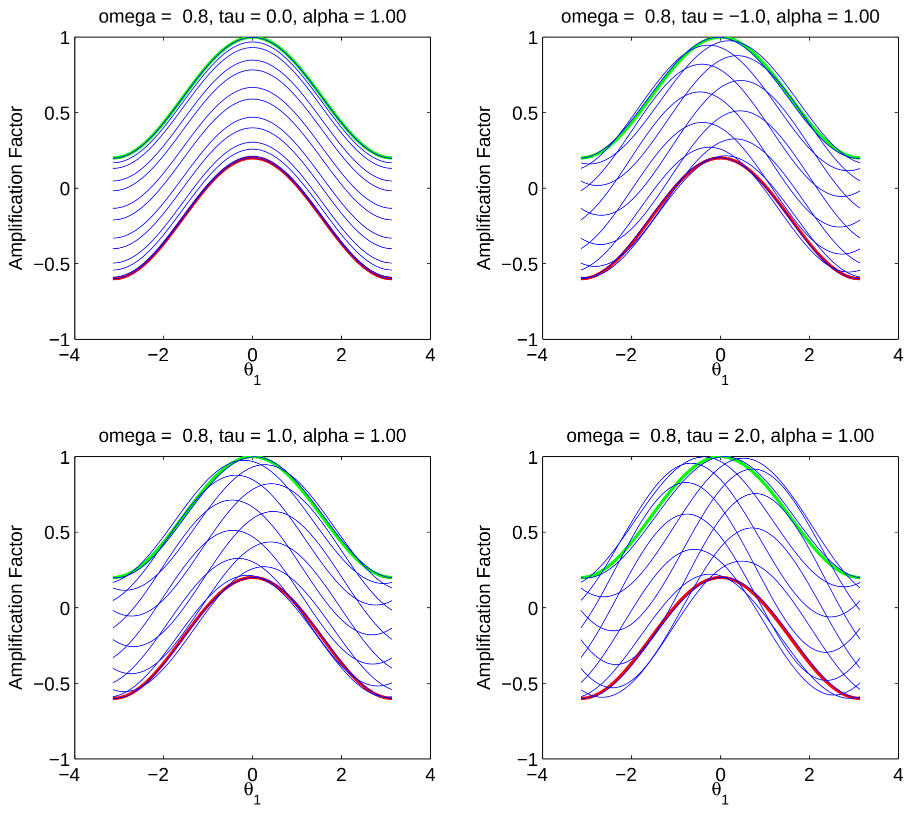


Figure 3: Amplification factor for damped Jacobi relaxations with $\omega = 0.8$ and $\alpha = h_x/h_y = 1$ and $\tau = -1, 0, 1, 2$ displayed as curves of constant θ_2. $\theta_2 = 0$ on the *green* curve and π on the *red* curve.

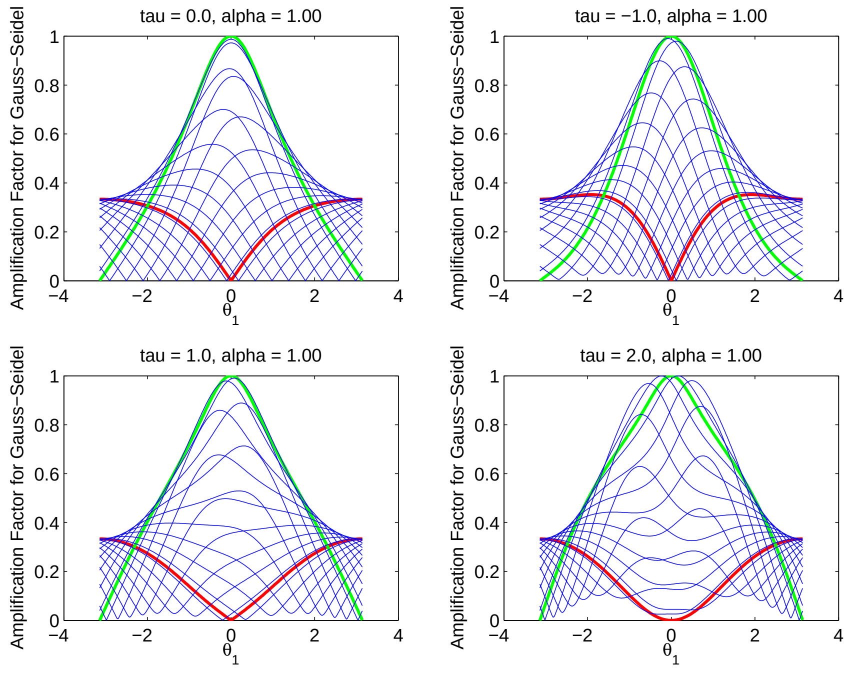


Figure 4: Absolute value of the amplification factor for Gauss-Seidel relaxations with $\alpha = h_x/h_y = 1$ and $\tau = -1, 0, 1, 2$, displayed as curves of constant θ_2. $\theta_2 = 0$ on the *green* curve and π on the *red* curve.

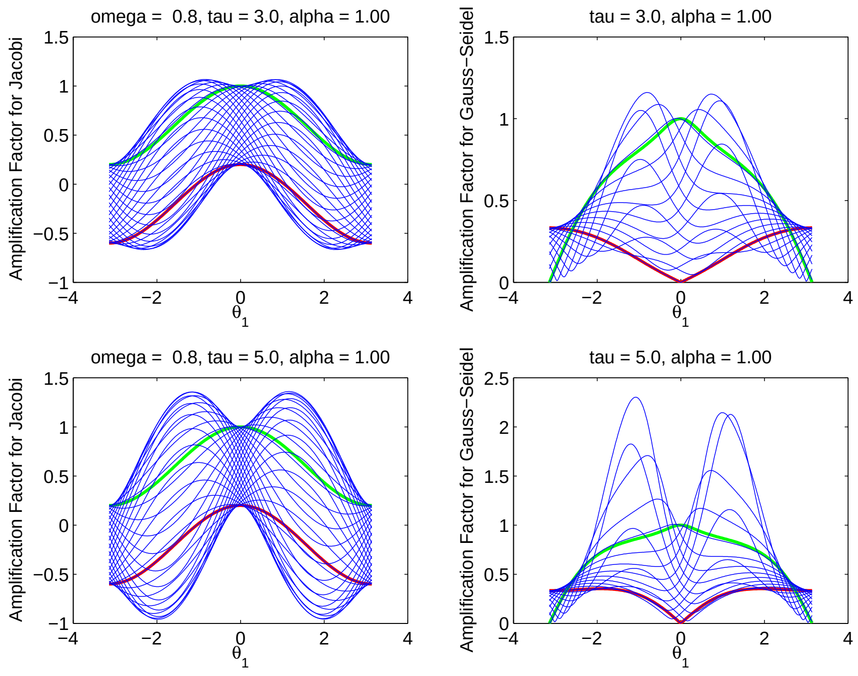


Figure 5: Amplification factor for Jacobi (left) and Gauss-Seidel (right) relaxations for $|\tau| = 3, 5$, $\alpha = h_x/h_y = 1$, displayed as curves of constant θ_2. $\theta_2 = 0$ on the *green* curve and π on the *red* curve.

4 Numerical Experiments

In the following numerical experiments, we look at the convergence rate of the residual norm and the error norm which are defined at the iteration m by

$$\begin{aligned} r^{(m)} &= \|\mathbf{f} - \mathbf{A}\mathbf{u}^{(m)}\|_\infty, \\ e^{(m)} &= \|\mathbf{u}^{(m)} - \mathbf{u}_{anal}\|_\infty. \end{aligned} \tag{24}$$

The iterations are stopped when the number of iterations reach an user supplied *maximum* of iterations or when the residual norm is smaller than either a given *relative* tolerance `rtol` [4, p. 51] or *absolute* tolerance `atol`:

$$\begin{aligned} r^{(m)} &< \texttt{rtol} \cdot (\|\mathbf{A}\|_\infty \cdot \|\mathbf{u}^{(m)}\|_\infty + \|\mathbf{f}\|_\infty), \\ r^{(m)} &< \texttt{atol}. \end{aligned} \tag{25}$$

An additional stopping criteria consists of stopping the iterations when the change of the discretization error between successive iteration is small enough:

$$\frac{e^{(m)} - e^{(m-1)}}{e^{(m-1)}} < \texttt{etol}. \tag{26}$$

4.1 V-cycle performances

Table 2 shows the numbers of V-cycles required to reach the *relative tolerance* `rtol` $= 10^{-8}$. In these runs where $\alpha = 0.5$, $\tau = 1$ and $a(x, y)$ given by Eq. 10, we observe that the biggest improvement is obtained at $\nu_1 = \nu_2 = 2$. For larger ν_1, ν_2, the number of required iterations is relatively insensitive to the grid sizes. As can be seen in Fig. 6 which plots the evolution of the error $e^{(m)}$, it is clear that the level of discretization error has been largely reached. Finally the times used by these runs are shown in Fig. 7 and Fig. 8 where the times spent in the direct solver using MUMPS [2] are included for comparison. For the 512×2048 grid (the largest case using the direct solver), the multigrid $V(3, 3)$ is about 30 times faster!

	`DDDD` problem				`NNDD` problem			
Grid size	$V(1,1)$	$V(2,2)$	$V(3,3)$	$V(4,4)$	$V(1,1)$	$V(2,2)$	$V(3,3)$	$V(4,4)$
16×64	3	2	2	1	4	2	2	1
32×128	5	3	2	2	5	3	2	2
64×256	7	4	3	3	7	4	3	3
128×512	10	6	4	4	10	6	5	4
256×1024	11	6	5	4	11	6	5	4
512×2048	11	6	5	4	11	6	5	4
1024×4096	10	6	4	4	10	6	4	4
1536×6144	9	6	4	4	9	5	4	3

Table 2: Multigrid V-cycle results for the `DDDD` and `NNDD` model problems with $k_x = k_y = 4$, $L_x = 100$, $L_y = 800$, $\tau = 1$ and $a(x, y)$ given by Eq. 10. Shown are the numbers of multigrid V-cycles required to reduce the *relative* residual norm to less than 10^{-8} for different grid sizes and numbers of pre and post relaxation sweeps. Gauss-Seidel relaxation is used. The coarsest grid size of the 1536×6144 case is 3×12 while all the others have a coarsest grid of size 2×8.

The fittings of the obtained data show that the multigrid V cycle cost scales almost *linearly* with the number of unknowns $N = (N_x + 1)(N_y + 1)$ (as does the backsolve stage of MUMPS) while the *total* direct solve time scales as $N^{1.4}$.

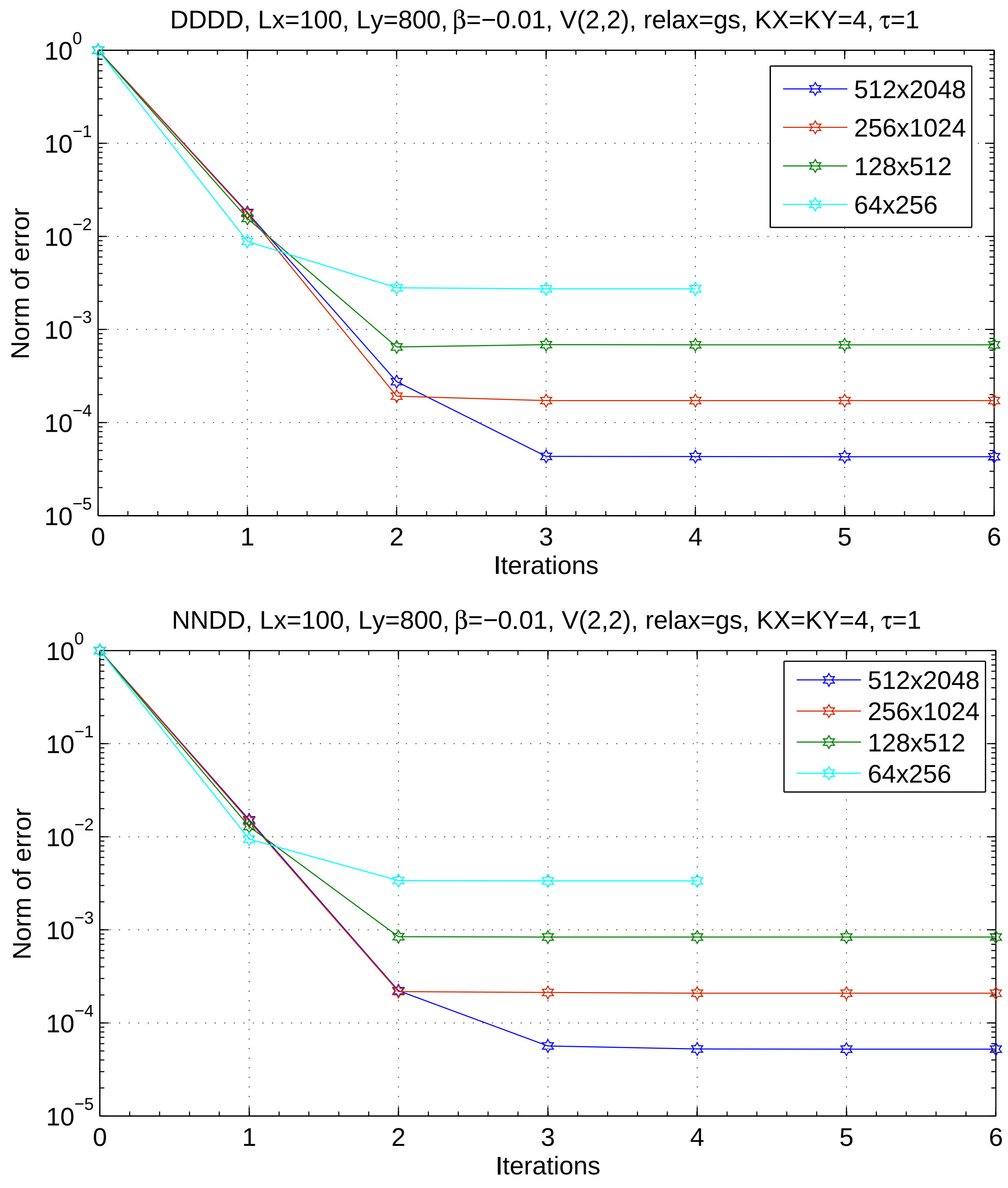


Figure 6: Performance of the $V(2,2)$-cycle using the Gauss-Seidel relaxation scheme for the `DDDD` (upper curve) and `NNDD` (lower curve) problem. The relative tolerance `rtol` is set to 10^{-8} the coarsest grid size for all the problem size is fixed to 2×8.

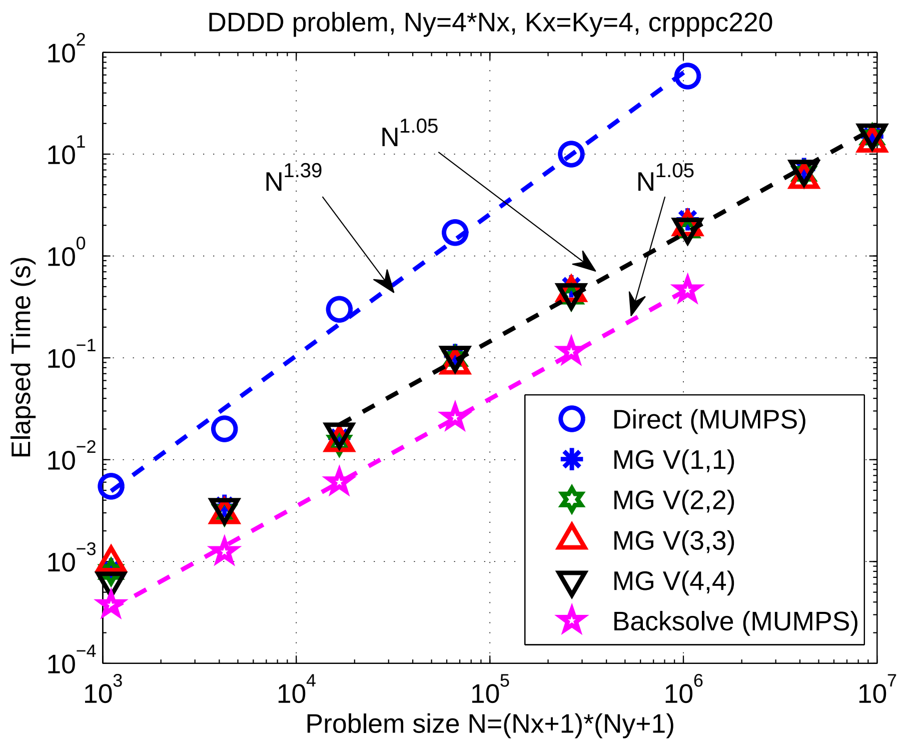


Figure 7: Times used by the multigrid V cycles for the runs reported in Table 2 for the DDDD problem. The last 6 $V(3,3)$ data points are used for the multigrid fit. The MUMPS direct solver's cost is included for comparison. All the timing results are obtained on an Intel Nehalem i7 processor, using the Intel compiler version 13.0.1 and MUMPS-4.10.0.

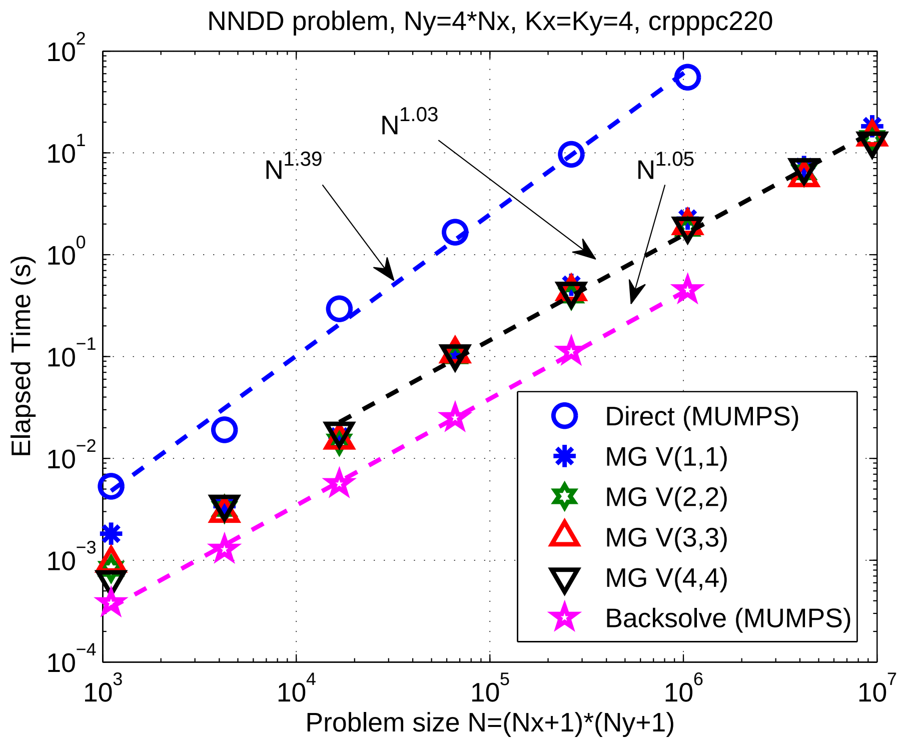


Figure 8: As in Fig. 7 for the NNDD problem.

4.2 Effects of the mesh aspect ratio α

From Table 3, one can observe that the required number of $V(2,2)$ cycles increase quickly when $\alpha < 0.5$ and $\alpha > 2$. Advanced *relaxation* methods and *coarsening* strategies [5, chap. 7] can solve this performance degradation but are generally more difficult to parallelize.

α	DDDD	NNDD
0.125	19	22
0.25	12	12
0.5	6	6
1.0	5	5
2.0	7	7
4.0	20	19

Table 3: Effects of the *mesh aspect ratio* $\alpha = h_x/h_y$ on the number of $V(2,2)$ cycles required to reach `rtol` $= 10^{-8}$ for DDDD and NNDD model problems. The listed α's are obtained by fixing $N_x = 256$, $N_y = 1024$, $L_x = 100$ and varying L_y.

4.3 Effects of the mixed partial derivative

When $|\tau| > 2$, Table 4 shows that the multigrid V-cycle diverge, as predicted from the local mode analysis based on the amplification factor given in Eq. 23. Although, a non-negative coefficient a has a *stabilizing* effect, the latter disappears already for a 256×1024 grid.

	Grid size	$\tau=-3$	$\tau=-2$	$\tau=-1$	$\tau=0$	$\tau=1$	$\tau=2$	$\tau=3$
DDDD	$128 \times 512 (a=0)$	-	39	7	5	7	38	-
	128×512	16	6	5	4	4	6	17
	256×1024	-	8	5	4	5	7	-
	512×2048	-	9	5	4	5	9	-
NNDD	$128 \times 512 (a=0)$	-	42	7	5	7	41	-
	128×512	13	6	5	4	5	5	13
	256×1024	-	7	5	4	5	7	-
	512×2048	-	7	5	4	5	7	-

Table 4: Effects of the mixed derivative term τ on the performances of the $V(3,3)$ cycle. The dashes indicate that the V-cycle diverges. In theses runs, $a(x,y)$ is given by Eq. 10 except for the cases where it is set to 0. Notice the *stabilizing* effect of $a \neq 0$ for the 128×512 grid at $\tau = \pm 3$.

4.4 Using the damped Jacobi relaxation

The optimum Jacobi damping factor ω can be determined by minimizing the *smoothing factor* defined as the maximum amplification coefficient (21) restricted to the *oscillatory modes*:

$$\mu(\omega,\alpha,\tau) = \max_{(\theta_1,\theta_2)\in\Omega} |G(\theta_1,\theta_2,\omega,\alpha,\tau)|, \qquad \Omega = [|\theta_1| > \pi/2] \bigcup [|\theta_2| > \pi/2]. \tag{27}$$

Results from numerical computation of (27 are shown in Fig. 9. An analytic expression for $\tau = 0$ assuming $\alpha \leq 1$ is derived in [6, p. 119]:

$$\mu(\omega,\alpha,\tau=0) = \max\left(|1-2\omega|\,,\,\left|1 - \frac{\alpha^2}{1+\alpha^2}\omega\right|\right),$$
$$\mu_{\text{opt}} = \frac{2+\alpha^2}{2+3\alpha^2} \quad \text{at} \quad \omega_{\text{opt}} = \frac{2+2\alpha^2}{2+3\alpha^2}. \tag{28}$$

Notice that the smoothing factor increases as α departs from 1 and for increasing τ.

For Gauss-Seidel relaxation, the same numerical procedure applied to (23) yields a smoothing factor μ equal to respectively 0.5, 0.68 and 0.70 for the three cases shown in Fig. 9, which result in a better smoothing property than the damped Jacobi relaxation.

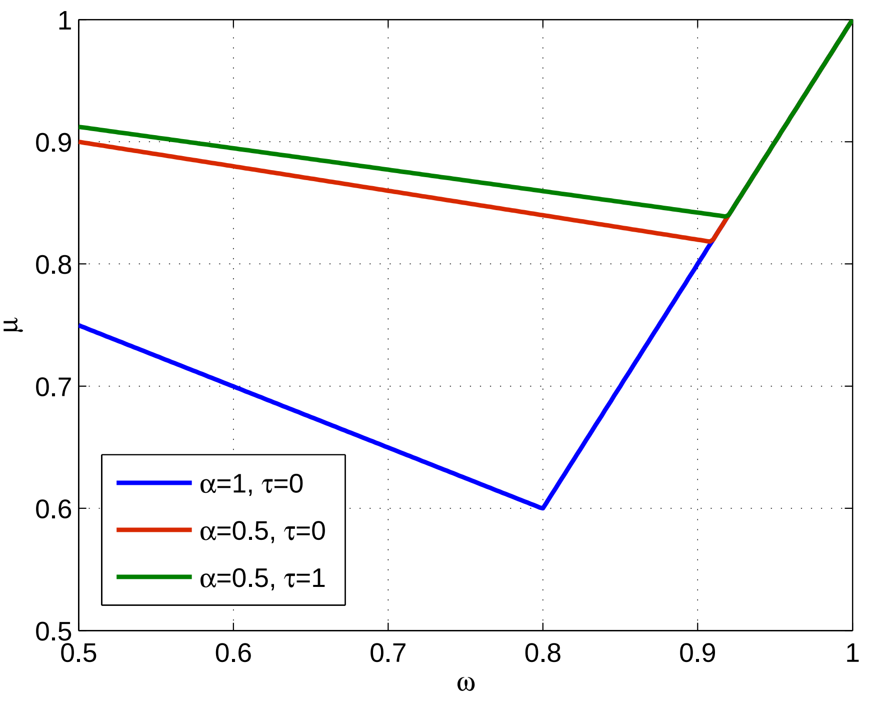


Figure 9: The smoothing factor for damped Jacobi relaxation for different values of α and τ.

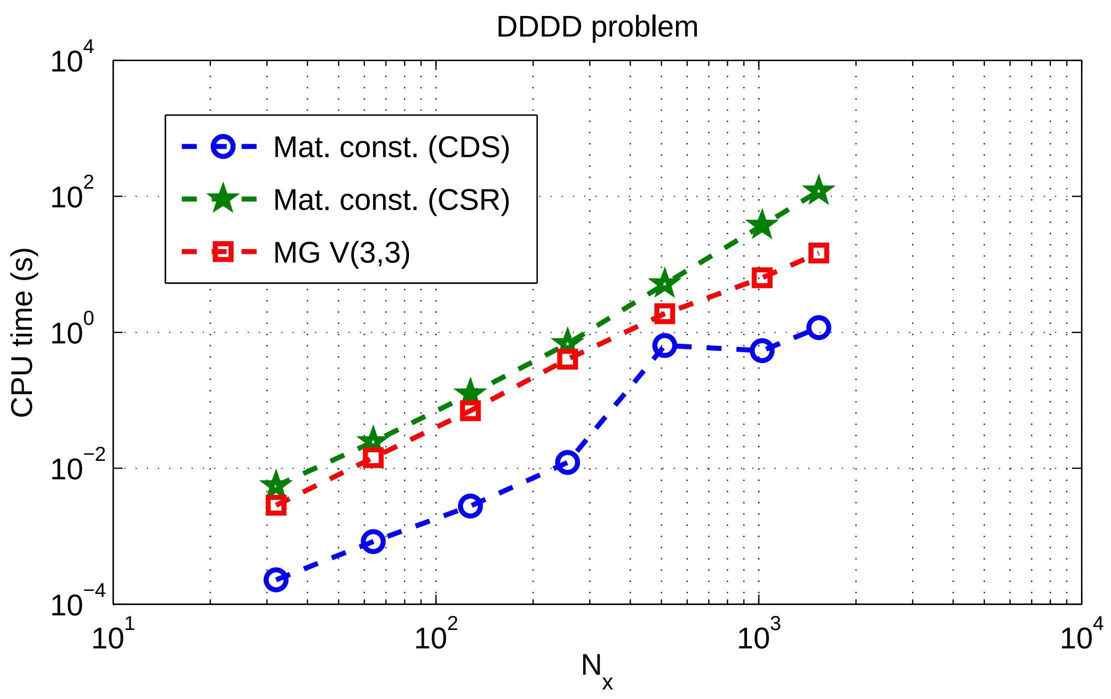
Numerical experiments with the reference case ($\alpha = 0.5$, $\tau = 1$, $a(x, y)$ given by Eq. 10) and the 128×512 grid using damped Jacobi relaxation, are shown in Table 5 and confirm that $\omega = 0.9$ is the optimum damping factor and that it is less efficient than the Gauss-Seidel relaxation, in agreement with the Fourier analysis.

	$\omega = 0.5$	$\omega = 0.6$	$\omega = 0.7$	$\omega = 0.8$	$\omega = 0.9$	$\omega = 1.0$
`DDDD`	12	10	9	8	7	15
`NNDD`	12	11	9	8	7	18

Table 5: The number of $V(3, 3)$ cycles required to obtain `rtol` $= 10^{-8}$ versus the Jacobi *damped factor* ω. The grid size is 128×512 with $\alpha = 0.5$, $\tau = 1$ and $a(x, y)$ given by Eq. 10.

4.5 Matrix storage

Initially the *Compressed Sparse Row* storage format (CSR or CRS) (see [4, p. 58–59]) was used to store the discretized finite difference matrix. With this choice, the CPU time used by the matrix construction (and boundary condition setting) is found to be always larger than the multigrid solver time as shown in Fig. 10. Fortunately, switching to the *Compressed Diagonal Storage* (CDS), where the 9 diagonal structure of the matrix is fully exploited, the matrix construction time is considerably reduced as shown in the same figure. On the other hand, no difference in the multigrid solver performance is noticeable between the two matrix storage.



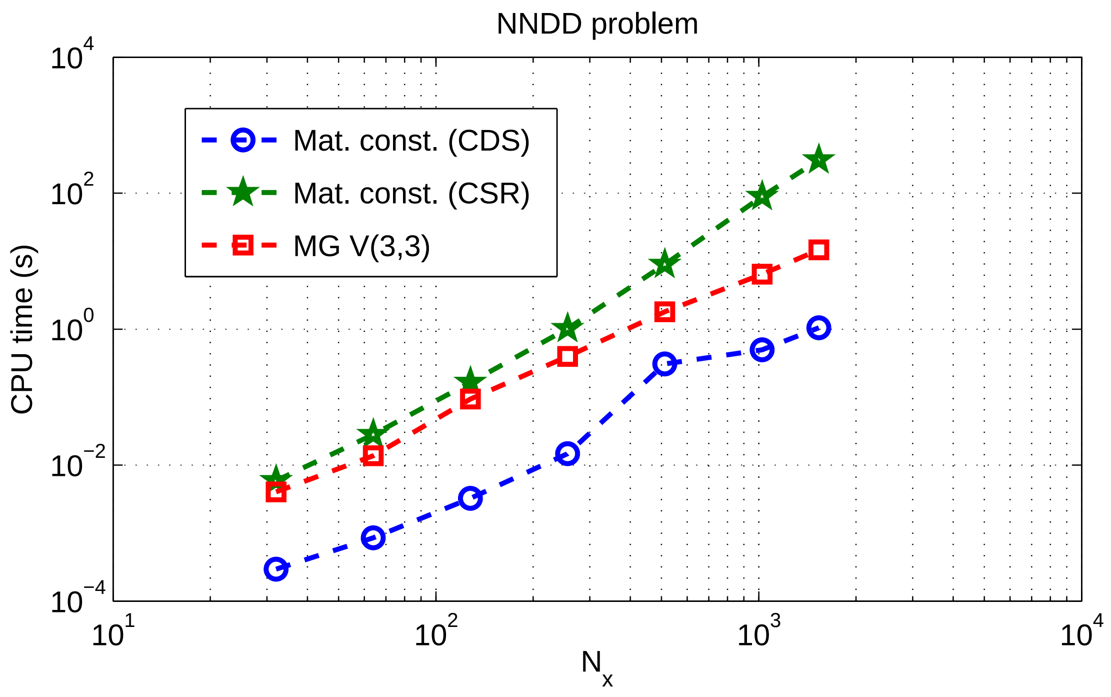


Figure 10: CPU time used by the matrix construction for CSR and CDS matrix storage compared to the multigrid $V(3,3)$ cycle time for the **DDDD** and **NNDD** model problems. The timing is obtained using the same conditions as in Fig. 7.

5 Modified PDE

Here, the following modified PDE is considered:

$$\left[\frac{\partial^2}{\partial x^2} + \tau\frac{\partial^2}{\partial x\partial y} + (1+\tau^2/4)\frac{\partial^2}{\partial y^2} - a(x,y)\right] u(x,y) = f(x,y), \qquad 0 \le x \le L_x,\ 0 \le y \le L_y. \tag{29}$$

This PDE which is obtained from the $\hat{s} - \alpha$ model [6, Eq.10] is *elliptic* for any value of τ. The resulting stencil is changed from the previous stencil 3 to

$$S_{ij} = \frac{1}{h_x^2}\begin{bmatrix} -\tau\alpha/4 & \alpha^2(1+\tau^2/4) & \tau\alpha/4 \\ 1 & -2\left[1+\alpha^2(1+\tau^2/4)\right] - h_x^2 a_{ij} & 1 \\ \tau\alpha/4 & \alpha^2(1+\tau^2/4) & -\tau\alpha/4 \end{bmatrix}. \tag{30}$$

Note that the *anisotropy* of the resulting Finite Difference discretization is now $\alpha^2(1+\tau^2/4)$ and could be controlled by adjusting both the mesh aspect ratio α and the *shear* term τ.

Numerical calculations show that the multigrid V-cycles do always converge, as shown in Table 6.

	Grid size	$\tau = 0$	$\tau = 1$	$\tau = 2$	$\tau = 4$	$\tau = 8$	$\tau = 16$
DDDD	128×512	4	4	5	6	9	20 (6)
	256×1024	4	4	5	6	11	25 (8)
	512×2048	4	4	5	7	12	29 (8)
NNDD	128×512	4	4	4	5	8	17 (5)
	256×1024	4	5	5	5	8	19 (6)
	512×2048	4	4	4	5	7	18 (6)

Table 6: Effects of the mixed derivative term τ on the performances of the $V(3,3)$ cycle. In theses runs, $a(x,y)$ is given by Eq. 10. The mesh aspect ratio $\alpha = 0.5$ was used. On the last column and shown in parenthesis are the numbers of $V(3,3)$ cycles when α is reduced to 0.125 by increasing the length L_y while keeping all the other values fixed.

6 Parallel Multigrid

In order to maximize the parallel efficiency and the flexibility of utilization, a two-dimensional domain partition scheme is chosen to parallelize the multigrid solver. As shown below, generalization of this procedure for higher dimensions is straightforward.

6.1 Distributed grid coarsening

The coarsening algorithm can be summarized as follow:

- Partition the grid points on each dimension at the *finest* grid level, as evenly as possible.
- The range for each sub-grid, using *global indexing* is thus specified by $[s, e]$, with $s = 0$ for the first sub-grid and $e = N$ for the last sub-grid, N being the number of grid intervals.
- The next coarse sub-grid is thus obtained by discarding all the *odd* indexed grid points, as in the serial case.
- This process can continue (as long as the total of number of intervals is even) until there exists a prescribed *minimum* number of grid points on any sub-grid is reached.

6.2 Matrix-free formulation

Using standard *matrix* to represent the discretized 2D (or higher dimensional) operators imply an *one-dimensional numbering* of the grid nodes. For example on a 2D $N_x \times N_y$ grid, the 1D numbering of the node (x_{i_1}, y_{i_2}) could be defined as

$$k = i_1 + i_2 \times N_x, \quad i_1 = 0 : N_x,\ i_2 = 0 : N_y.$$

However, using 2D domain partition defined by

$$i_1 = s_1 : e_1,\ i_2 = s_2 : e_2, \tag{31}$$

with $s = (s_1, s_2)$ and $e = (e_1, e_2)$ denoting respectively the *starting* and *ending* indices of a rectangular sub-domain, result in a *non-contiguous* set of the indices k and in a complicate structure of the partitioned matrix for the linear operator.

On the other hand, using the *stencil notation* introduced in [5, chap. 5.2] based on the *multidimensional* node labeling as defined by (31) for a 2D problem, one can define a simple data structure for the partitioned operator, $A(i, \delta)$, where the d-tuple $i = (i_1, \ldots, i_d)$ represents a node of the d-dimensional grid and the d-tuple $\delta = (\delta_1, \ldots, \delta_d)$, the *distance* between the connected nodes. The result of $\mathbf{A}u$ can thus be defined as

$$(\mathbf{A}u)_i = \sum_{\delta \in \mathbb{Z}^d} A(i, \delta) u_{i+\delta}, \quad i = s : e. \tag{32}$$

In (32), the sum is performed over all indices δ such that $A(i, \delta)$ is non-zero. For the 2D nine-point stencil defined in (3), the 2-tuple δ can be specified as the 9 columns of the following *structure* matrix

$$S_\delta = \begin{pmatrix} 0 & -1 & 0 & 1 & -1 & 1 & -1 & 0 & 1 \\ 0 & -1 & -1 & -1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{33}$$

In the general case of a d-dimensional grid and $\mathcal{N}$ point stencil, S_δ is a $d \times \mathcal{N}$ matrix. By noting that the subscript $i + \delta$ of u on the right hand side of (32) should be in the range $[0, N]$ *only* for sub-domains which are *adjacent* to the boundary, one can deduce that for a *fixed* δ, the lower and upper bounds of the indices i should be

$$\begin{aligned} i_{\min} &= \max(0, -\delta, s), \\ i_{\max} &= \min(N, N - \delta, e) \end{aligned} \tag{34}$$

where $N = (N_1, N_2, \ldots, N_d)$ specify the number of intervals, since, for sub-domains *not adjacent* to the boundary, u should include values at the *ghost* cells $s - g$ and $e + g$ where g is given by

$$g = \max |S_\delta| \tag{35}$$

with the operator max taken along the *rows* of the matrix. The formula defined in (32) can then be implemented as in the *pseudo* Fortran code

```
do k=1,SIZE($S_\delta$,2)  ! loop over the stencil points
   $\delta$ = $S_\delta$(:,k)
   lb = MAX(0,$-\delta$,$s$)
   ub = MIN($N$,$N-\delta$,e)
   do i=lb,ub
      Au(i) = Au(i) + A(i,$\delta$)*u(i+$\delta$)
   enddo
enddo
```

On the other hand, if the values of u at the ghost cells of the sub-domains *adjacent* to the boundary are set to 0

$$u_{-g} = u_{N+g} = 0,$$

the lower and upper bounds of the inner loop can be simply set to $lb = s$ and $ub = e$. Note that the inner loop should be interpreted as d nested loops over the d-tuple $i = (i_1, \ldots, i_d)$ for a d-dimensional problem.

6.3 Inter-grid transfers

6.3.1 Restriction

Using the definition in the first equation of (13) together with (12), the 2D restriction operator can be represented by the following 9-point stencil:

$$\mathbf{R}_i = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}, \tag{36}$$

and the restriction of f can be computed as

$$\bar{f}_i = (\mathbf{R}f)_i = \sum_{\delta\in\mathbb{Z}^2} R(i,\delta) f_{2i+\delta}, \quad i = \bar{s} : \bar{e}, \tag{37}$$

where $\bar{s}, \bar{e}$ denote the partitioned domain boundary indices on the *coarse* grid, using the same algorithm described previously.

6.3.2 BC for the restriction operator

Dirichlel boundary conditions can be imposed by modifying the *restriction stencil* on each of the four boundaries as follow:

$$\mathbf{R}_{0,\cdot} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 0 \\ 2 & 4 & 0 \\ 1 & 2 & 0 \end{pmatrix}, \quad \mathbf{R}_{N_x,\cdot} = \frac{1}{16}\begin{pmatrix} 0 & 2 & 1 \\ 0 & 4 & 2 \\ 0 & 2 & 1 \end{pmatrix}, \quad \mathbf{R}_{\cdot,0} = \frac{1}{16}\begin{pmatrix} 0 & 0 & 0 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}, \quad \mathbf{R}_{\cdot,N_y} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 0 & 0 & 0 \end{pmatrix}. \tag{38}$$

With the natural Neumann BC, no change of the restriction operator is needed.

6.3.3 Prolongation

Stencil notation for *prolongation* operators is less obvious to formulate, see [5, chap. 5.2]. A more straightforward implementation is obtained in the 2D case, by simply applying *bilinear interpolation* on the *coarse grid*:

$$\begin{aligned} (\mathbf{P}\bar{u})_{2i} &= \bar{u}_i, \\ (\mathbf{P}\bar{u})_{2i+e_1} &= (\bar{u}_i + \bar{u}_{i+e_1})/2, \quad (\mathbf{P}\bar{u})_{2i+e_2} = (\bar{u}_i + \bar{u}_{i+e_2})/2, \\ (\mathbf{P}\bar{u})_{2i+e_1+e_2} &= (\bar{u}_i + \bar{u}_{i+e_1} + \bar{u}_{i+e_2} + \bar{u}_{i+e_1+e_2})/4, \end{aligned} \tag{39}$$

6.4 Relaxations

While the Gauss-Seidel proves to be more efficient, the damped Jacobi method, at least for a first version of the parallel multigrid solver, is used because it is straightforward to *parallelize*. The same undamped Jacobi (with $\omega = 1$) with a *few* number of iterations is also used to solve the linear system at the coarsest mesh as prescribed by the multigrid V-cycle procedure defined in section 3.

6.5 Local vectors and stencils

All local vectors (used to represent solutions or right-hand-sides) contain *ghost cells* and are implemented using 2D arrays, for example

```
sol(s(1)-1:e(1)+1,s(2)-1:e(2)+1)
```

for the solution vector.

The partitioned stencils are defined only for the *local* grid points, without the ghost cells. Thus, before each operation on the local vectors, an exchange (or update) of the values on the ghost cells is performed.

As a result, all the memory required by the solver is completely partitioned, except for the space used by the ghost cells.

6.6 Numerical Experiments

In this section, all the numerical experiments are conducted on `helios.iferc-csc.org`, using the Intel compiler version 13.1.3 and bullxpmi-1.2.4.3. The *stopping criteria* for the V-cycles is based on the absolute and relative residual norms as well as the discretization error norm as defined in section 4. In cases where the analytic solution is not known, the latter can be replaced by some norm of the solution.

6.6.1 Strong scaling

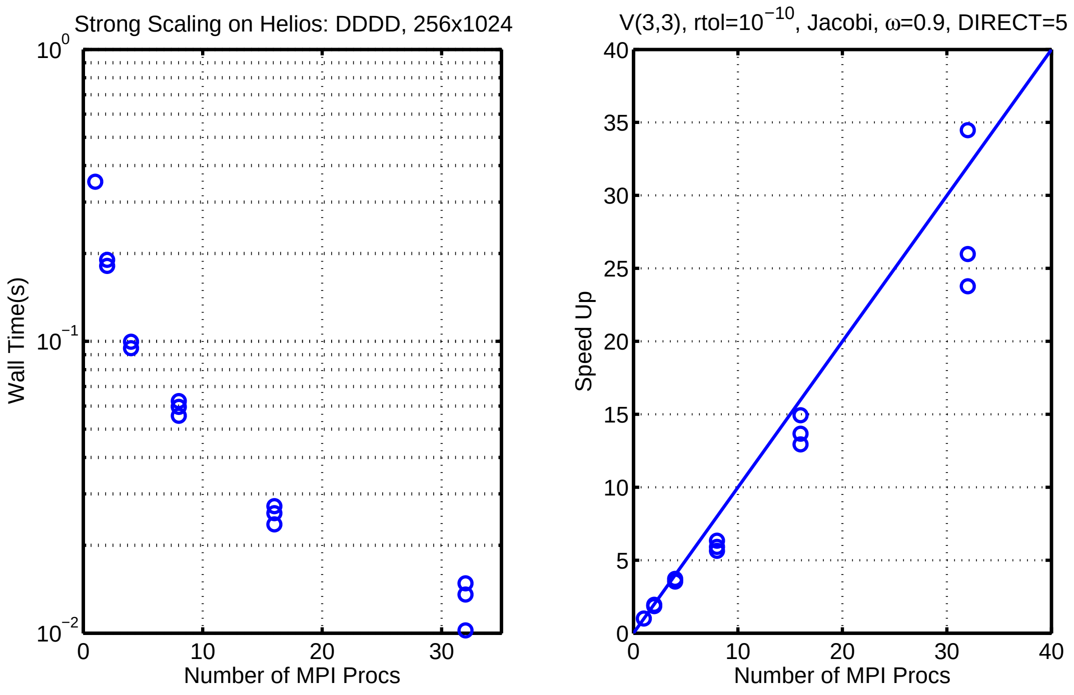


Figure 11: DDDD problem for a 256×1024 size, using multigrid $V(3,3)$ cycles. Different times for a given number of processes are obtained with different combinations of processes in each dimension. The number of grid levels are fixed to 6. Five Jacobi iterations are used at the coarsest grid.

Here 2 *fixed* problem sizes are considered:

- A small size with the (fine) grid of 256×1024 shown in Fig. 11 and
- a larger size of 512×2048 in Fig. 12.

In both cases `rtol` $= 10^{-8}$ and `etol` $= 10^{-3}$. It was checked that the results do not change when more than 5 Jacobi iterations are used at the coarsest mesh. Notice that for the small problem, the parallel efficiency starts to degrade at 32 MPI processes while for the larger case, this happens after 64 MPI processes. This can be explained by the ghost cell exchange communication overhead: denoting N_1 and N_2, the number of

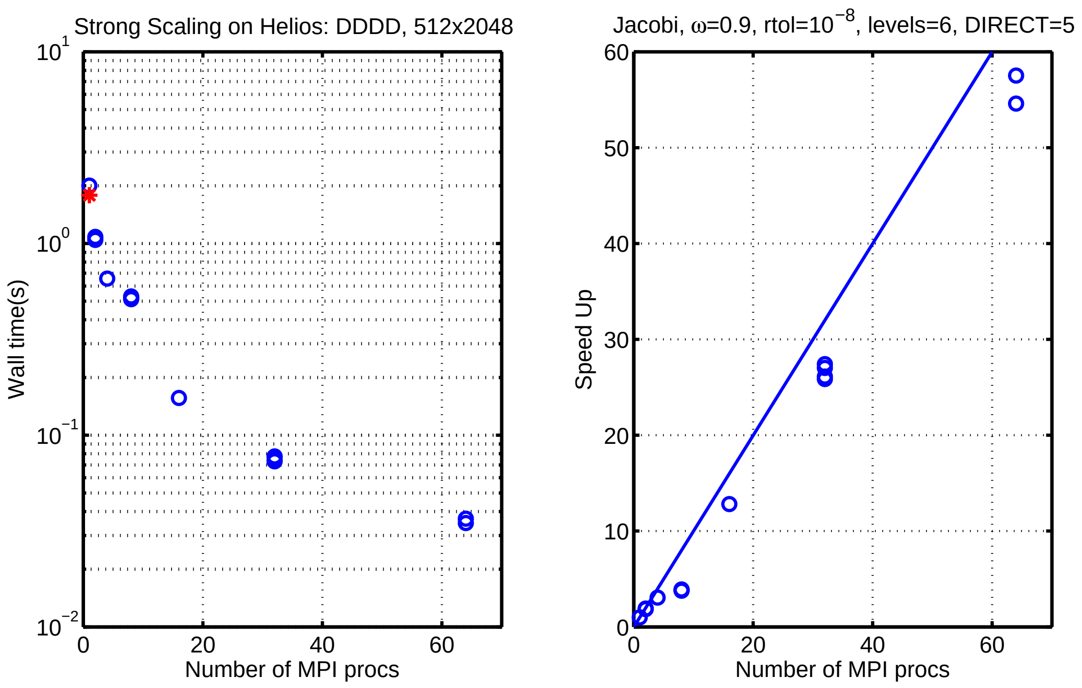


Figure 12: DDDD problem for a 512×2048 size using multigrid $V(3,3)$ cycles. The red marker on the left shows the time for the serial multigrid solver. Different times for a given number of processes are obtained with different combinations of processes in each dimension. The number of grid levels are fixed to 6. Five Jacobi iterations are used at the coarsest grid.

grid points in each direction and P_1 and P_2 the number of MPI processes in each direction, the ratio S/V between the number of ghost points and interior grid points for each local subdomains can be estimated as

$$S/V \simeq \frac{2(N_1/P_1 + N_2/P_2)}{N_1 N_2/P_1 P_2} = 2\,(P_1/N_1 + P_2/N_2)\,. \tag{40}$$

This ratio increases as the number MPI processes increases while keeping the problem size fixed. On very coarse grids, this communication cost can become prohibitive. For this reason, in all the runs shown here, the number of grid points on each direction for the coarsest grid is limited to 2.

6.6.2 Weak Scaling

According to Eq. 40, varying the problem size together with the number of MPI processes by keeping N_1/P_1 and N_2/P_2 constant should yield a *constant scaling*, provided that the convergence rate does not depend on the problem sizes. The results for the `DDDD` and `NNDD` problems are shown in Fig. 13 and Fig. 14. The left part of the figures shows that the convergence rate depends only weakly on the problem sizes, which leads indeed to a (almost) constant time obtained for numbers of MPI processes P between 16 and 1024 . The reason for the good timings for smaller P is simply that there are only 2 ghost cell exchanges for $P = 2 \times 2$ (instead of 4 for $P \geq 16$) and that there is no exchange for $P = 0$.

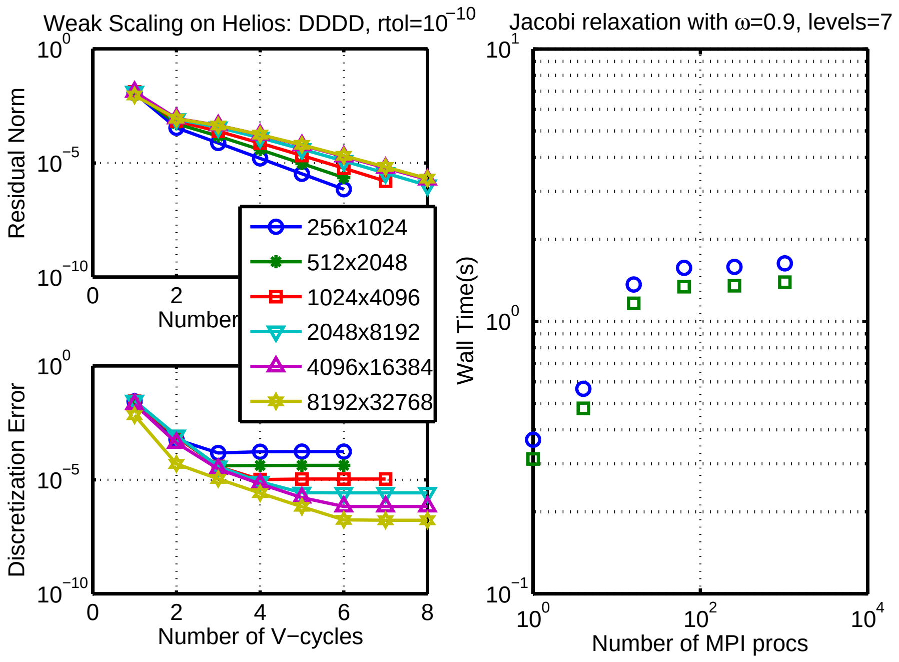


Figure 13: Weak scaling for a DDDD problem, using multigrid $V(3,3)$ cycles. The number of grid levels are fixed to 7. The solver for the coarsest grid uses 5 Jacobi iterations except for the 2 largest cases which require respectively 20 and 100 iterations to converge. The 2 sets of curves on the right figure show respectively the timings with and without the calculations of the residual norm and discretization error which require both a *global reduction*.

7 Non-homogeneous Boundary Conditions

7.1 Non-homogeneous Dirichlet Conditions

Non-homogeneous Dirichlet boundary conditions can be imposed on all the Dirichlet faces simply by *clearing*, as for the *homogeneous case*, the matrice rows and columns and setting its diagonal term to 1. Moreover, the corresponding corresponding *right-hand-side* should be set to:

$$\begin{aligned} f_{0,j} &= D^W(y_j), \quad f_{N_x,j} = D^E(y_j), \qquad j = 0,\ldots,N_y, \\ f_{i,0} &= D^S(x_i), \quad f_{j,N_y} = D^N(x_i), \qquad i = 0,\ldots,N_x, \end{aligned} \tag{41}$$

where D^W, D^E, D^S, D^N are the values of u at the 4 Dirichlet faces. As for the homogeneous Dirichlet BC, the *restriction* operator should be changed as described in section 6.3.2 while the *prolongation* defined in (39) remains unchanged.

7.2 Non-homogeneous Neumann Conditions

The non-homogeneous Neumann conditions at the 4 faces $x = 0$ can be defined as

$$\begin{aligned} \left.\frac{\partial u}{\partial x}\right|_{x=0} &= N^W(y), \quad \left.\frac{\partial u}{\partial x}\right|_{x=L_x} = N^E(y), \\ \left.\frac{\partial u}{\partial y}\right|_{y=0} &= N^S(x), \quad \left.\frac{\partial u}{\partial y}\right|_{y=L_y} = N^N(x). \end{aligned} \tag{42}$$

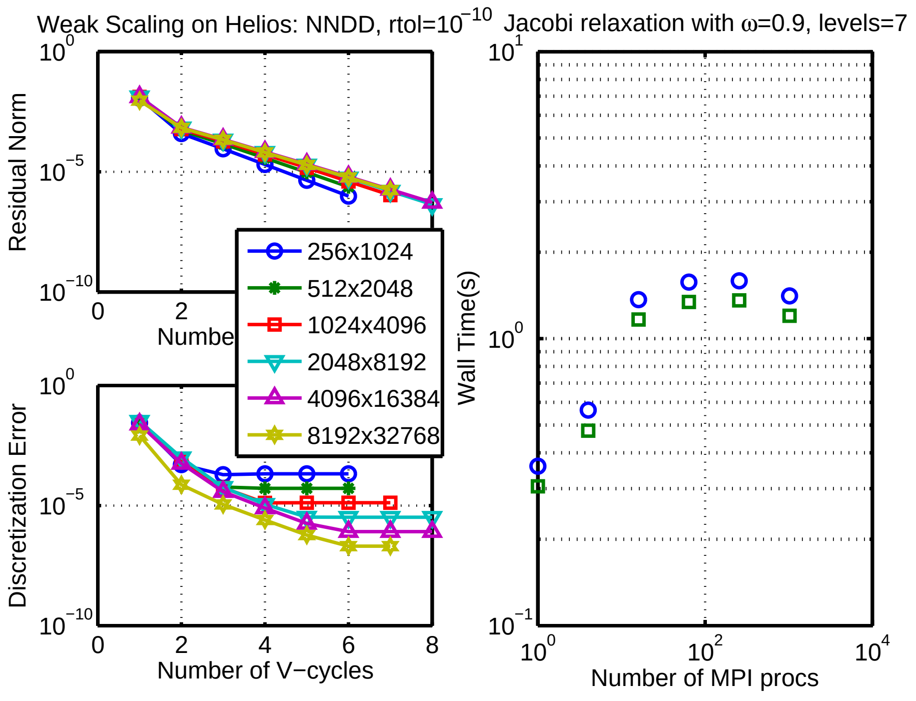


Figure 14: Weak scaling for a NNDD problem, using multigrid $V(3,3)$ cycles. The number of grid levels are fixed to 7. The solver for the coarsest grid uses 5 Jacobi iterations except for the 2 largest cases which require respectively 20 and 100 iterations to converge. The 2 sets of curves on the right figure show respectively the timings with and without the calculations of the residual norm and discretization error which require both a *global reduction*.

Discretization of the BC defined above, using the *central difference* yields on the 4 faces

$$\begin{aligned} u_{-1,j} &= u_{1,j} - 2h_x N^W(y_j), \quad u_{N_x+1,j} = u_{N_x-1,j} + 2h_x N^E(y_j), \qquad j = 0,\dots,N_y, \\ u_{i,-1} &= u_{i,1} - 2h_y N^S(x_i), \quad u_{i,N_y+1} = u_{i,N_y-1} + 2h_y N^N(x_i), \qquad i = 0,\dots,N_x. \end{aligned} \tag{43}$$

With these relations, the stencil (30) on the 4 boundaries is modified as follow

$$\begin{aligned} S^W &= \frac{1}{h_x^2}\begin{bmatrix} 0 & \alpha^2(1+\tau^2/4) & 0 \\ 0 & -2(1+\alpha^2(1+\tau^2/4)) - h_x^2 a_{0,j} & 2 \\ 0 & \alpha^2(1+\tau^2/4) & 0 \end{bmatrix}, \quad S^E = \frac{1}{h_x^2}\begin{bmatrix} 0 & \alpha^2(1+\tau^2/4) & 0 \\ 2 & -2(1+\alpha^2(1+\tau^2/4)) - h_x^2 a_{N_x,j} & 0 \\ 0 & \alpha^2(1+\tau^2/4) & 0 \end{bmatrix}, \\ S^S &= \frac{1}{h_x^2}\begin{bmatrix} 0 & 2\alpha^2(1+\tau^2/4) & 0 \\ 1 & -2(1+\alpha^2(1+\tau^2/4)) - h_x^2 a_{i,0} & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad S^N = \frac{1}{h_x^2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2(1+\alpha^2(1+\tau^2/4)) - h_x^2 a_{i,N_y} & 1 \\ 0 & 2\alpha^2(1+\tau^2/4) & 0 \end{bmatrix}, \end{aligned} \tag{44}$$

while the right-hand-side should be changed according to

$$\begin{aligned} f_{0,j} &\longleftarrow f_{0,j} + \frac{2}{h_x}\left[\frac{\tau\alpha}{4}N^W(y_{j-1}) + N^W(y_j) - \frac{\tau\alpha}{4}N^W(y_{j+1})\right], \\ f_{N_x,j} &\longleftarrow f_{N_x,j} + \frac{2}{h_x}\left[\frac{\tau\alpha}{4}N^E(y_{j-1}) - N^E(y_j) - \frac{\tau\alpha}{4}N^E(y_{j+1})\right], \\ f_{i,0} &\longleftarrow f_{i,0} + \frac{2}{h_y}\left[\frac{\tau}{4\alpha}N^S(x_{i-1}) + (1+\tau^2/4)N^S(x_i) - \frac{\tau}{4\alpha}N^S(x_{i+1})\right], \\ f_{i,N_y} &\longleftarrow f_{i,N_y} + \frac{2}{h_y}\left[\frac{\tau}{4\alpha}N^N(x_{i-1}) - (1+\tau^2/4)N^N(x_i) - \frac{\tau}{4\alpha}N^N(x_{i+1})\right]. \end{aligned} \tag{45}$$

7.3 The NNDD test problem

In order to test the discretization of the non-homogeneous boundary conditions as formulated above, a test problem with the prescribed *exact* solution

$$u(x,y) = 1 + \sin\frac{2\pi k_x x}{L_x}\sin\frac{2\pi k_y y}{L_y}, \qquad \text{where } k_x,\ k_y \text{ are positive integers} \tag{46}$$

and the following non-homogeneous boundary conditions

$$\begin{aligned} \left.\frac{\partial u}{\partial x}\right|_{x=0} &= \left.\frac{\partial u}{\partial x}\right|_{x=L_x} = k_x \sin\frac{2\pi k_y y}{L_y}, \\ u(x,0) &= u(x,L_y) = 1, \end{aligned} \tag{47}$$

is solved with varying grid spacing. The discretization errors versus the number of grid intervals N_x displayed in Fig(15 shows a *quadratic* convergence as expected from the second order finite differences used in both the PDE and the Neumann boundary condition discretization.

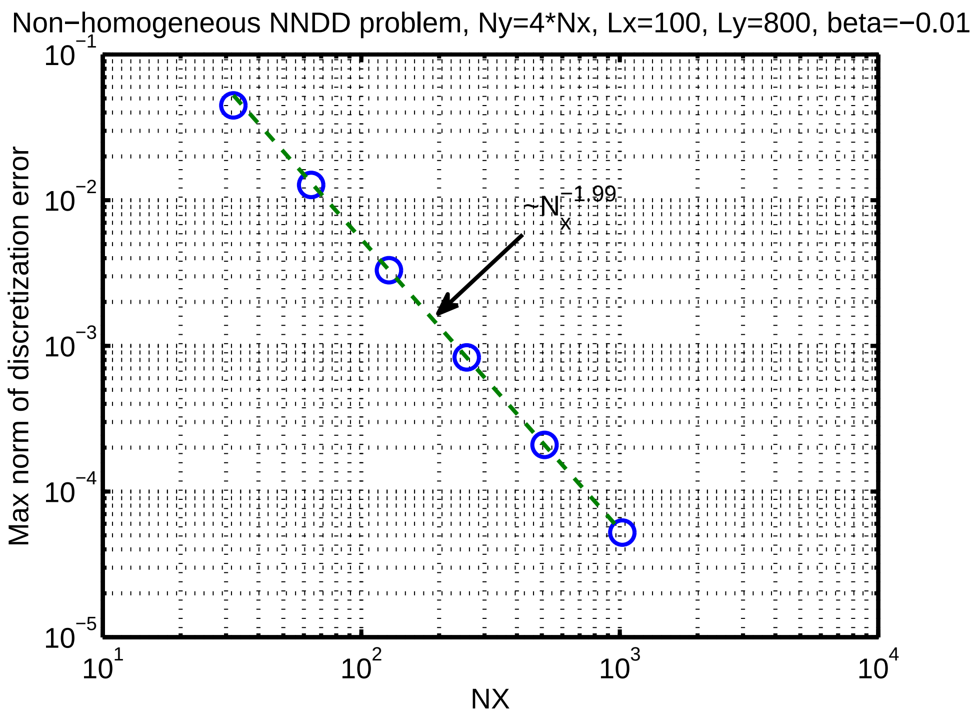


Figure 15: Convergence of the error $\|u_{calc} - u_{anal}\|_\infty$ wrt the number of intervals in the x direction N_x for the non-homogeneous NNDD problem. Here, $L_x = 100$, $L_y = 800$, $k_x = k_y = 4$, $\tau = 1$ and $N_y = 4N_x$.

As shown in Fig.(16), the multigrid V-cycles for the *non-homogeneous* problem converge with a slightly smaller efficiency, than the *homogeneous* problem shown in Fig.(14).

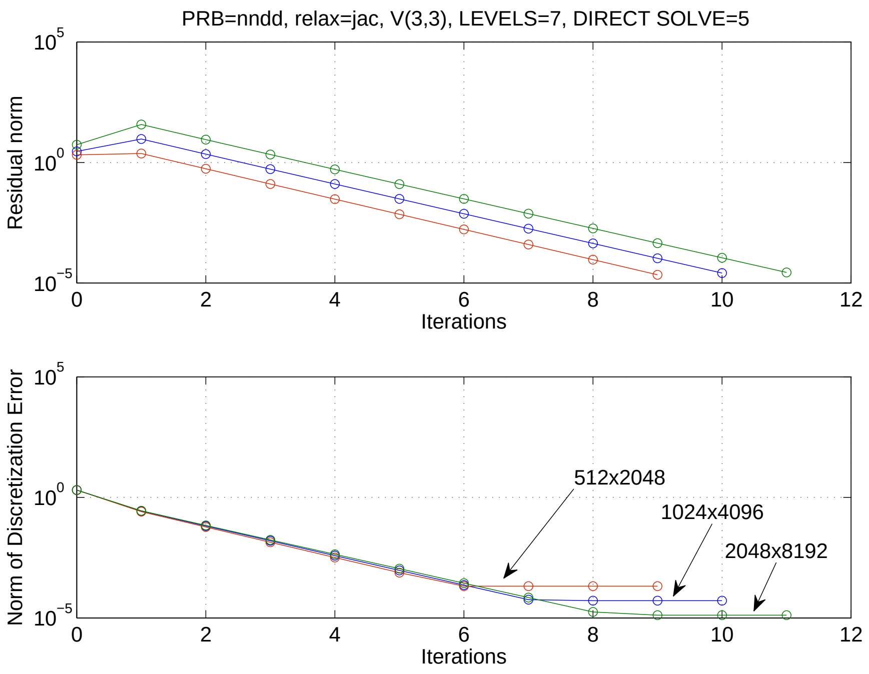


Figure 16: Performances of the $V(3,3)$-cycle for the non-homogeneous NNDD problem. The same parameters in Fig.(15) are used here.

7.4 Local relaxation methods

In addition to the damped Jacobi, three methods of relaxations are added in this parallel multigrid solver:

1. The 4 color Gauss-Seidel method (RBGS).
2. The Gauss-Seidel method (GS).
3. The successive over-relaxation method (SOR).

In order to apply correctly the parallel 4 color Gauss-Seidel, a complicated ghost cell exchange has to be performed for each sweep for each color. Here we simply apply the method *locally* on each subdomain with only one ghost exchange performed at the beginning of each relaxation.

The same procedure is also used for the other 2 methods which are inherently *serial.* All these 3 relaxations are thus only correct if there is only one subdomain. As a consequence, while the damped Jacobi does not depend on the partition of the subdomains, results from these 3 methods do depend on how the domain is partitioned.

Table 7 show however that all of the 3 *approximated* relaxation methods produce a much *faster* convergence rate than the damped Jacobi relaxations for the NNDD test problem considered here. The performance of the implemented solver using the 4 relaxation methods on HELIOS is compared in Fig,(17. The bad performance of the 4 color Gauss-Seidel relaxations (RBGS) can be explained by the 4 nested loops required to sweep each of the 4 colors.

Grid Sizes	256×1024	512×2048	1024×4096	2048×8192	4096×16384	8192×32768
Process topology	1×1	2×2	4×4	8×8	16×16	32×32
Jacobi $\omega = 0.9$	0.22	0.24	0.24	0.24	0.24	0.25
RBGS	0.05	0.07	0.10	0.10	0.12	0.12
GS	0.07	0.08	0.10	0.11	0.11	0.12
SOR $\omega = 1.2$	0.04	0.05	0.07	0.07	0.07	0.08

Table 7: Reduction factor for the residuals (obtained as the *geometric mean* of all its values except the first 2 values) for the non-homogeneous NNDD test problem. The same parameters as in Fig.(15) are used here.

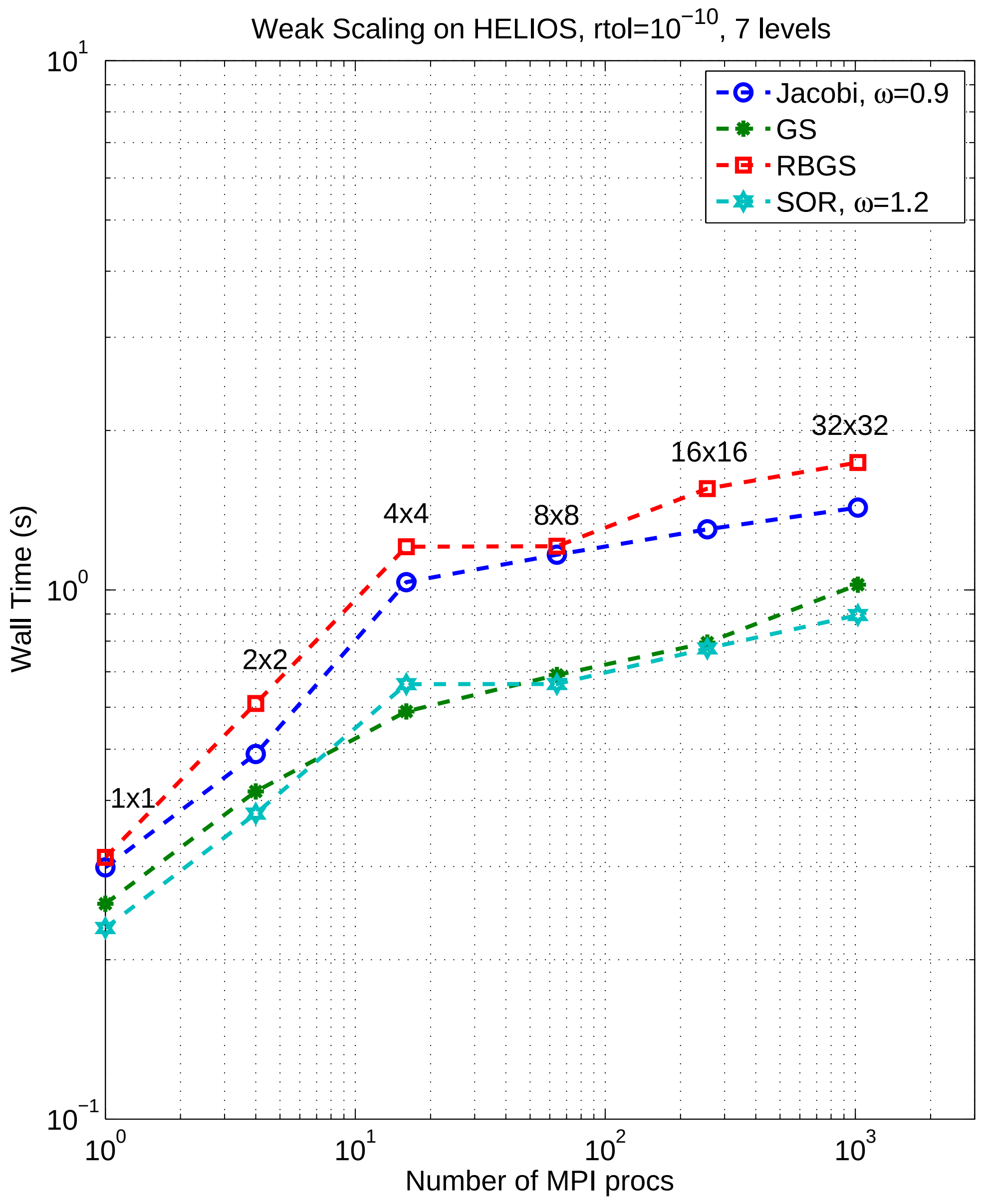


Figure 17: Performance of the 4 relaxations on the non-homogeneous NNDD problem. The same parameters in Fig.(15) are used here. The grid sizes used in this *weak scaling* run are shown in Table 7.

8 Performance of the Stencil Kernel on different platform

To get a feeling on the performances gained on the different platforms and how well the compilers (with their auto-vectorization capability) support these platforms, the following Fortran *9-point stencil* kernel has been used. The OpenMP directives are used for parallelization on both Xeon and Xeon Phi while offload to GPU card is done via the high level OpenACC directives. *First touch* is applied in the initialization of `x` and `mat`.

```
!$omp parallel do private(ix,iy)
!$acc parallel loop present(mat,x,y) private(ix,iy)
    DO iy=0,ny
        DO ix=0,nx
            y(ix,iy) =  mat(ix,iy,1)*x(ix-1,iy-1) &
                  &      + mat(ix,iy,2)*x(ix,  iy-1) &
                  &      + mat(ix,iy,3)*x(ix+1,iy-1) &
                  &      + mat(ix,iy,4)*x(ix-1,iy)   &
                  &      + mat(ix,iy,0)*x(ix,iy)     &
                  &      + mat(ix,iy,5)*x(ix+1,iy)   &
                  &      + mat(ix,iy,6)*x(ix-1,iy+1) &
                  &      + mat(ix,iy,7)*x(ix,  iy+1) &
                  &      + mat(ix,iy,8)*x(ix+1,iy+1)
        END DO
    END DO
!$acc end parallel loop
!$omp end parallel do
```

The performances on a Helios dual processor node and its attached Xeon Phi co-processor are shown in Fig. 18 while the performances on a Cray XC30 CPU and its attached NVIDIA graphics card are shown in Fig. 19. In these figures, Intel optimization flag `-O3` and default Cray optimization were applied. In Fig. 20, the speedup by vectorization is shown by comparing performances obtained with `-O3` and `-O1`. Several observations can be drawn from these results.

- The parallel scaling, using OpenMP is linear for both Intel and Cray compilers, when the problem sizes fit into the 20MB cache of the Sandybridge processor. For grid sizes smaller than 32×8, the overhead of thread creation dominates. When the memory footprint is larger than the cache, 4 threads per socket already saturate the memory bandwidth.
- On the MIC, the parallel speedup scales linearly up to 60 cores with 1 thread per core. Using 2 or 3 threads per core does not help while with 4 threads, the performance even degrades.
- The MIC, using the Intel *mic native* mode, does not perform better than 8 cores of the Sandybridge processor.
- Since the benefit from *vectorization* is quite large for the MIC (see Fig. 20), the poor parallel scalability may be explained by the low flop intensity per thread coupled with the high overhead of the (many) thread creation and thread synchronization.
- The NIVIDIA card, using the high level OpenACC programming style is more than 3 times faster than 8 Sandybridge cores, for grid sizes larger than 1024×256. For smaller sizes, there are not enough flops to keep the GPU threads busy.

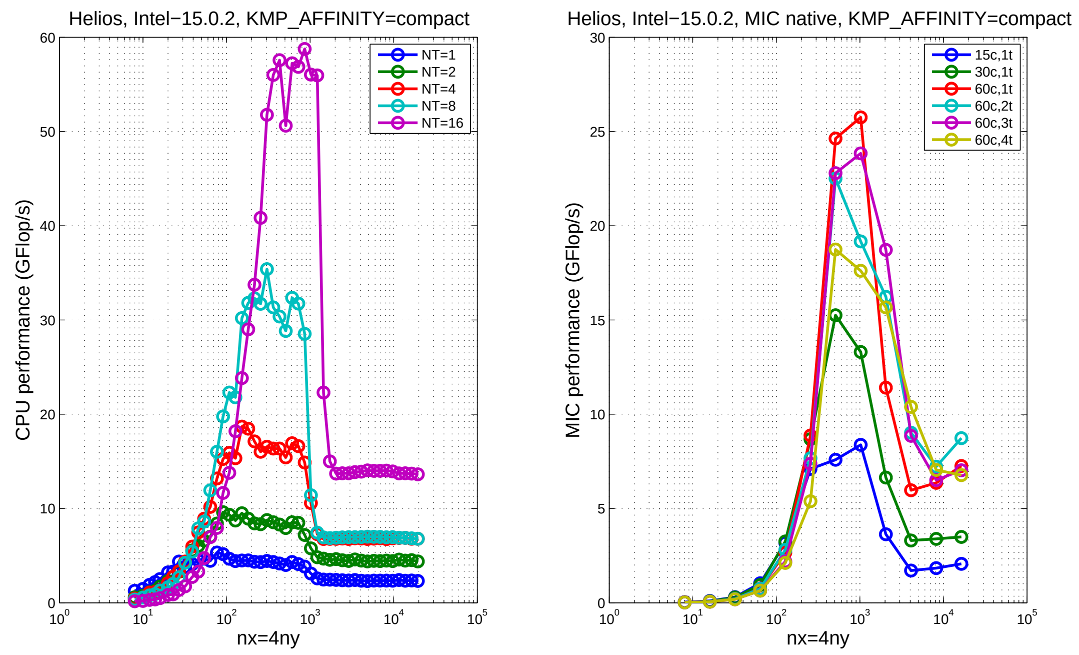


Figure 18: Performance on the Helios dual processor (left) using the `-O3` compiler option and on the MIC (right), using the native mode `-mmic`.

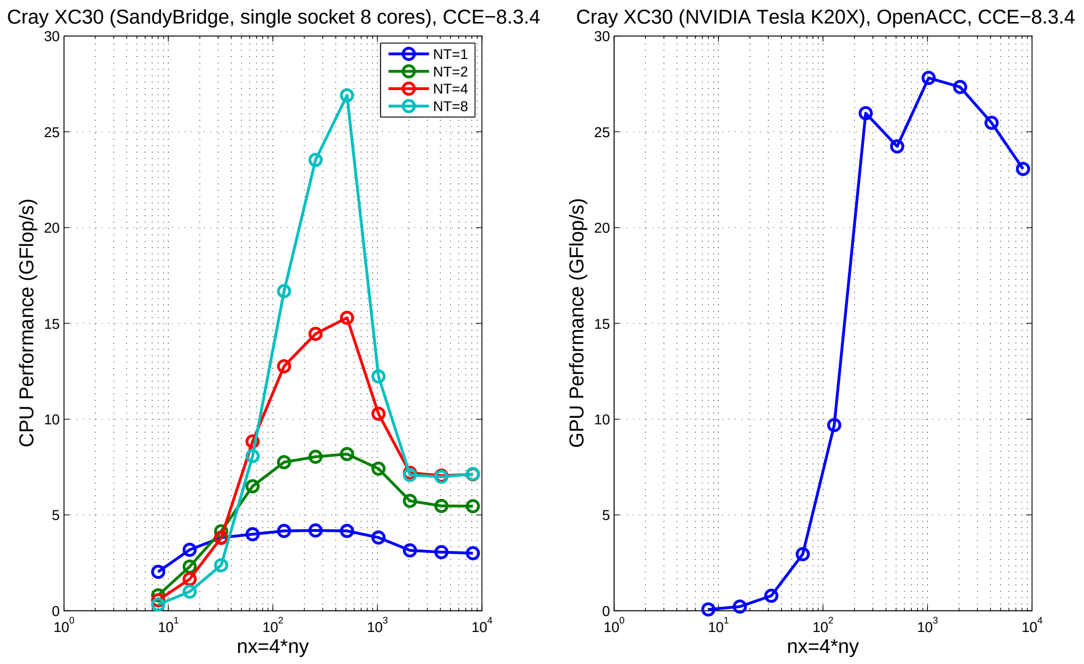


Figure 19: Performance on a Cray XC30 single 8 core processor node (left) and the NVIDIA card (right) using OpenACC. Default Cray Fortran compiler optimization has been used on both runs.

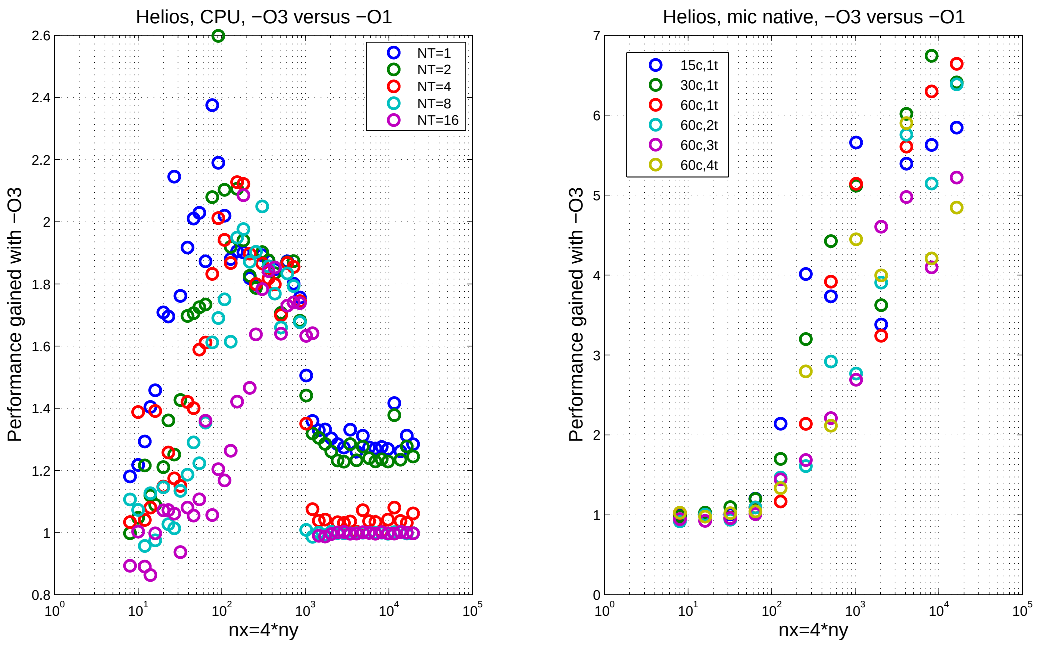


Figure 20: Performance comparison between using `-O3` and `-O1`, on the Helios dual processor (left) and on the MIC (right).

9 Hybrid MPI+OpenMP `PARMG (r599)`

In this version, a *straightforward* parallelization is done in the subroutines **`jacobi, residue, prolong, restrict`** and **`norm_vec`**, using the OpenMP work sharing directives. The ghost cell exchange is executed by the *master* thread. All the 2D arrays (solutions, RHS, etc.) are allocated and initialized *once* by the *master* thread. Dynamic array allocations during the multigrid V-cycles are thus avoided.

To help further optimization, timings are introduced for each of the 4 multigrid components **`jacobi, residue, prolong, restrict`** and the ghost cell **`exchange`** as well as on the *recursive* subroutine **`mg`**. Since the timings of the 4 MG components include already calls to **`exchange`**, the time obtained for **`mg`** should be equal to the sum of the 4 MG components and the *extras* time which includes operations in **`mg`** but not in the 4 components:

$$t_{\text{mg}} = t_{\text{jacobi}} + t_{\text{residue}} + t_{\text{prolong}} + t_{\text{restrict}} + t_{\text{extras}}. \tag{48}$$

We will see in the following sections that, in addition to these 5 contributions to t_{mg}, there is *overhead* probably due to the *recursive* calls of **`mg`**.

9.1 Parallel efficiency on single node

The comparison in Fig. 21 shows that the pure OpenMP version is at most 30% slower than the pure MPI version when all the 16 cores are used but less than 10% when only one socket is used. The degradation of the OpenMP version can be explained by the *numa* effects when 2 sockets are used. It is also observed that the performance level off at 4 cores, due to the saturation of the socket memory bandwidth.

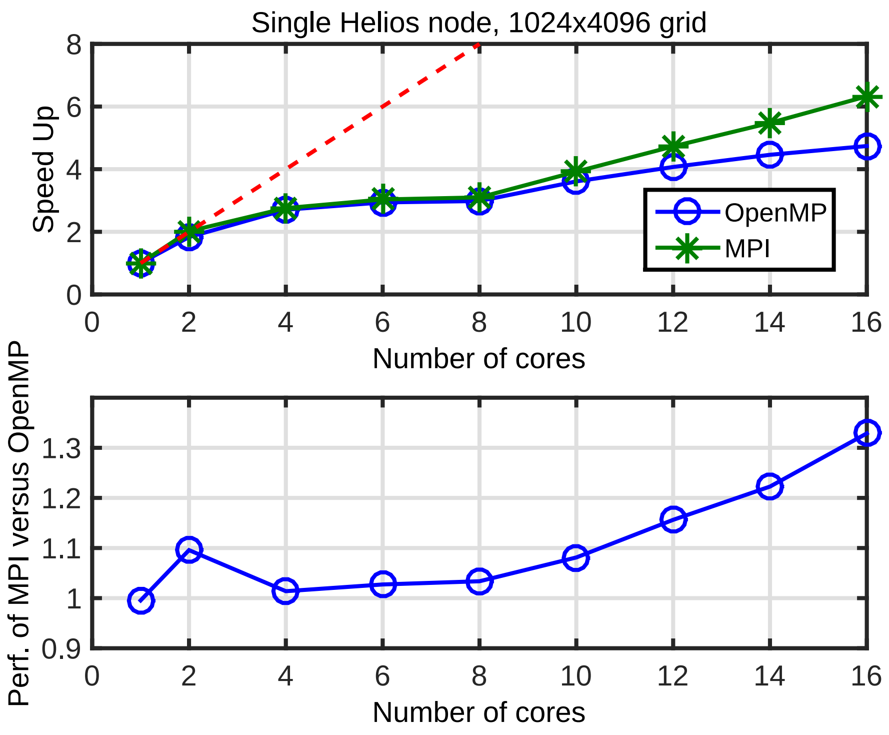


Figure 21: Parallel performance of the 7 level $V(3,3)$-cycle on a dual socket Helios node (2×8 cores) for pure OpenMP and pure MPI. The non-homogeneous NNDD problem with the same parameters in Fig. 15 is considered here. The OpenMP threads and MPI tasks are placed first on the first socket before filling the second socket, using the **`srun`** option "`--cpu_bind=cores -m block:block`" and the environment variable `OMP_PROC_BIND=true`.

9.2 Hybrid efficiency on multi-nodes

In the following multi-node experiments, all the 16 cores on each Helios node are utilized. The numbers of OpenMP threads *per* MPI *process* NT, the number of MPI processes *per node* NP, the number of nodes NNODES and the *total* number of MPI processes NP_{tot} verify thus the following relations:

$$\begin{gathered} 1 \leq \mathrm{NT} \leq 16, \qquad 1 \leq \mathrm{NP} \leq 16 \\ \mathrm{NT} \times \mathrm{NP} = 16 \\ \mathrm{NP}_{tot} = 16 \times \mathrm{NNODES}/\mathrm{NT} \end{gathered} \tag{49}$$

The times of the different MG components and the relative contributions for the *strong scaling* experiments using a 1024×4096 grid size, are shown in Fig. 22 and Fig. 23 respectively. The following observations can be made:

1. The `exchange` time increases strongly with increasing NNODES, due to smaller partitioned subdomains and thus their larger surface/volume ratio.
2. The pure MPI (NT=1) `exchange` time is on the other hand reduced with NT > 1 since the local partitioned grid becomes larger.
3. The less efficient OpenMP parallelization (numa effects, Amdahl's law) tends to limit however this advantage.
4. As a result, there is an optimal NT for a given NNODES: 2 for 4 and 16 nodes, 8 for 64 nodes.
5. The `jacobi` and `residue` contributions dominate largely with $0.63 \leq t_{\mathrm{jacobi}}/t_{\mathrm{mg}} \leq 0.83$ and $0.09 \leq t_{\mathrm{residue}}/t_{\mathrm{mg}} \leq 0.18$.
6. The *overhead* (see Eq. 48) times increase with NNODES but decrease slightly for increasing NT.

The times of the different MG components and the relative contributions for the *weak scaling* experiments are shown in Fig. 24 and Fig. 25 respectively. The following observations can be made:

1. A steady increase of MG times with the number of nodes can be attributed to the increase of ghost cells `exchange` time, even though the amount of communications between nodes does not change.
2. The MG performance is improved slightly when NT=2 but drop drastically for NT=16. This seems to indicate that *numa* effects are important here, since the array initialization is not done locally on each thread.
3. The *overhead* (see Eq. 48) times are much smaller than in the *strong scaling* runs.

Finally, Table 8 shows that using NT > 1 decreases the memory needed by the multigrid procedure for both strong and weak scaling runs.

9.3 Summary and conclusions

The *strong scaling* and *weak scaling* wrt NT and NNODES are summarized in Fig. 26. The speed up for the strong scaling experiments shows a good efficiency up to 16 nodes for all NT but degrades at 64 nodes (1024 cores) due the partitioned grid becoming too small. A good *weak scaling* is also obtained with an increase in t_{mg} of less than 10% when NNODES vary from 4 to 64. However, for NT=16, the efficiency drops significantly, due to the non-local memory access when the OpenMP threads are placed on both sockets (*numa* effect).

In order to improve the hybrid MPI+OpenMP multigrid, especially for large number of threads per MPI process NT, the following optimizations should be done:

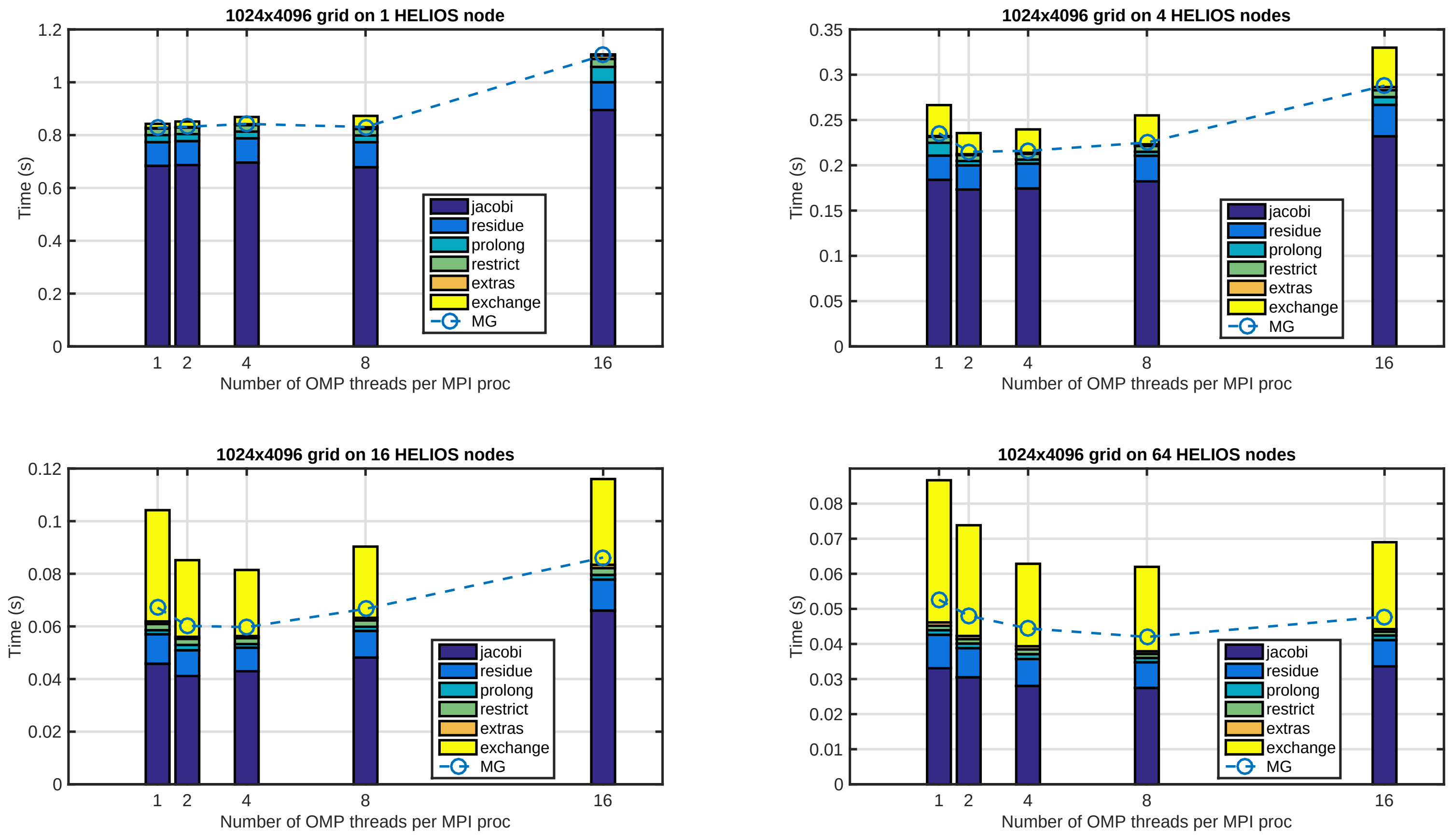


Figure 22: Detailed timings for strong scaling experiments using the same problem parameters as in Fig. 21, except that 5 levels are chosen to be able to run the runs with 64 nodes.

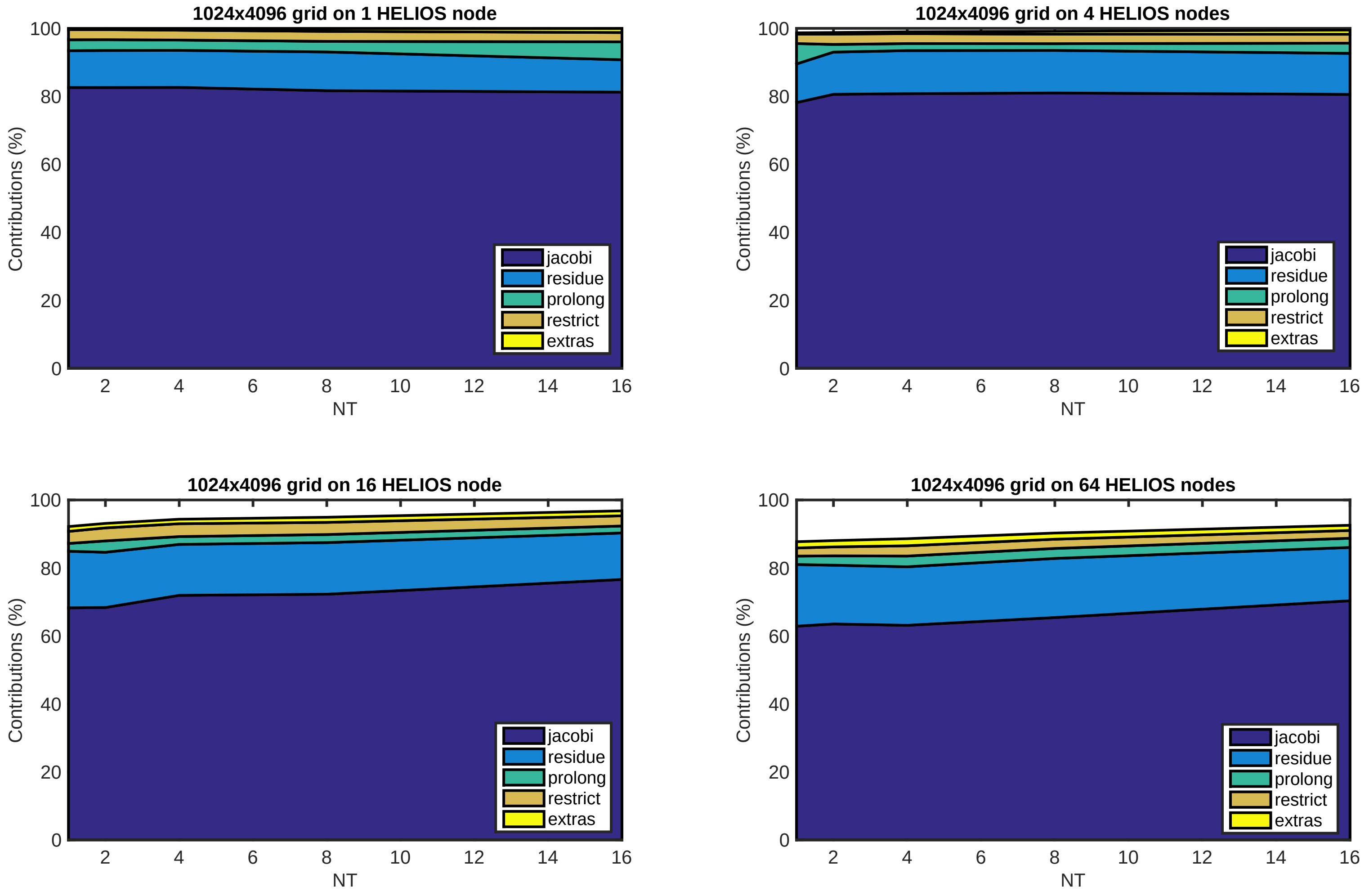


Figure 23: Relative contributions of each of the MG components for the strong scaling experiments using the same problem parameters as in Fig. 21.

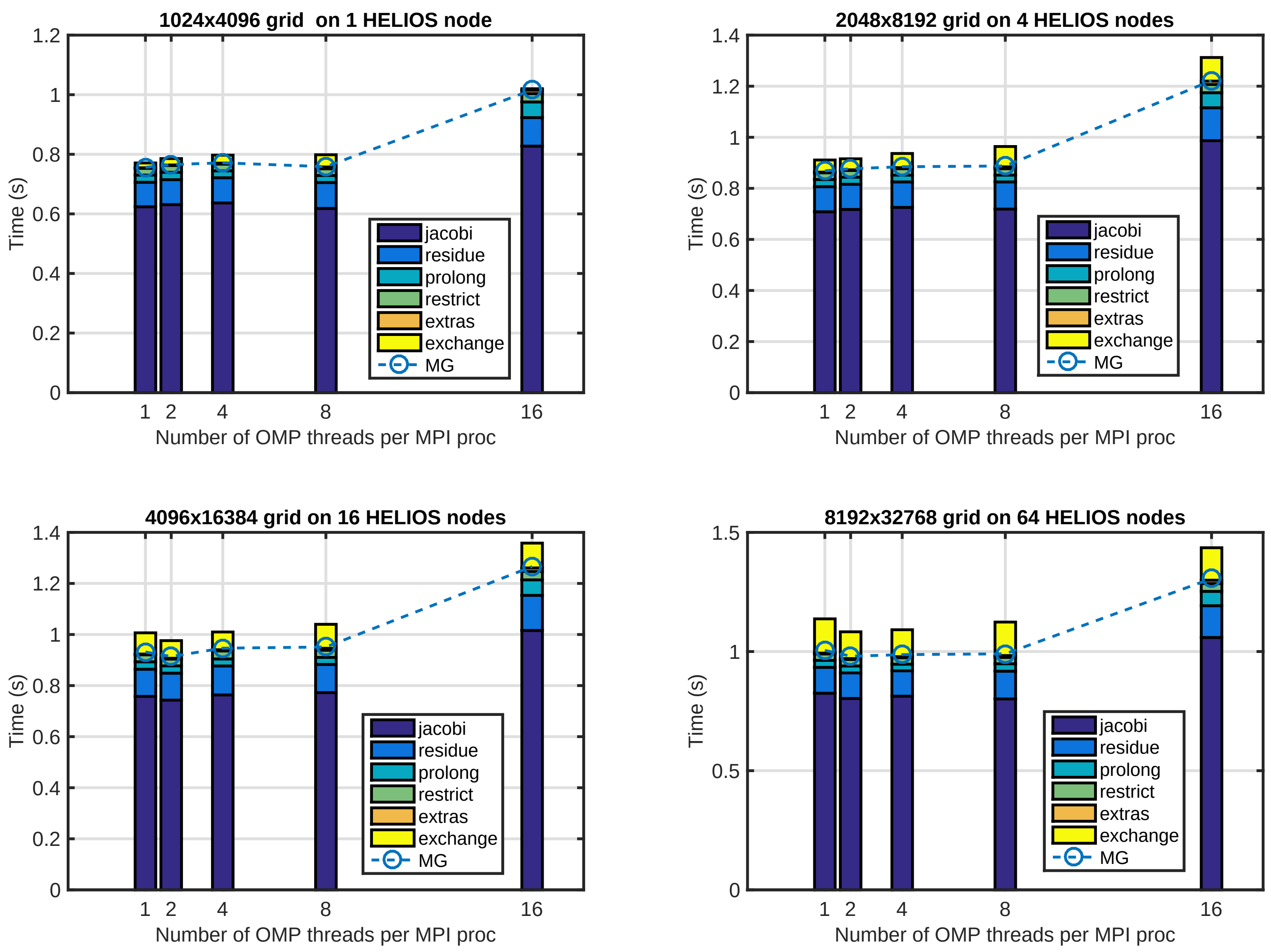


Figure 24: Detailed timings for weak scaling experiments using the same problem parameters as in Fig. 21.

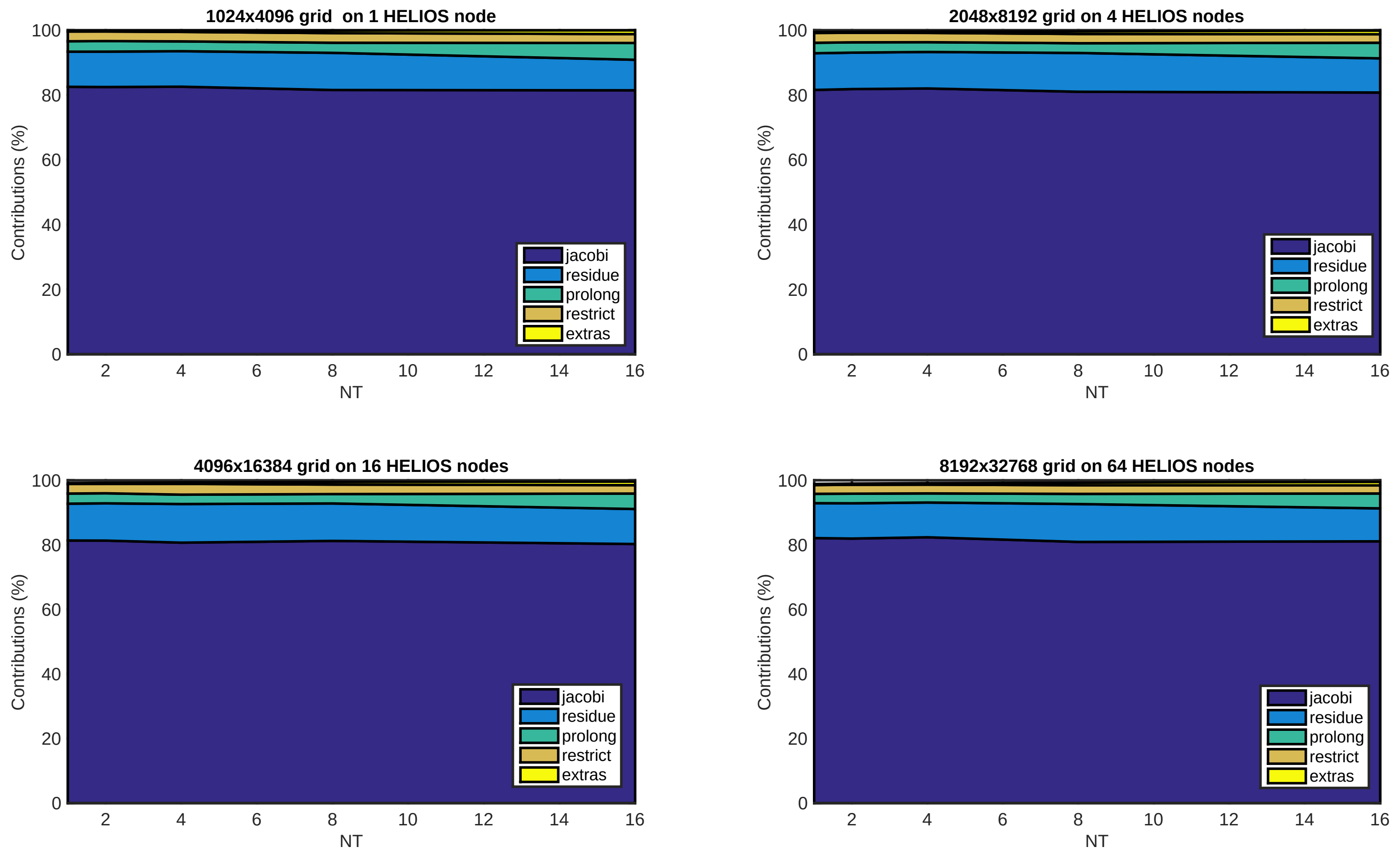


Figure 25: Relative contributions of each of the MG components for the weak scaling experiments using the same problem parameters as in Fig. 21.

	`NNODES`	NT=1	NT=2	NT=4	NT=8	NT=16
`Strong Scaling`	1	57.01	53.70	52.04	51.06	48.57
	4	21.87	21.49	15.75	13.87	13.11
	16	13.82	8.52	5.75	5.69	4.00
	64	13.93	6.80	3.73	2.32	1.48
`Weak Scaling`	1	57.03	53.71	52.04	51.08	48.61
	4	59.24	59.18	53.39	51.52	48.69
	16	60.48	55.33	52.69	52.74	49.06
	64	63.30	56.13	53.30	51.58	48.79

Table 8: Memory footprint *per core* (MB/core) for the strong scaling and weak scaling experiments.

- *First touch* array initialization in order to avoid *numa* effects.
- OpenMP parallelization of some remaining *serial* loops.
- Better vectorization of inner loops.

The outcome of these optimization steps is important in order to run efficiently on upcoming *multicore* processors and *manycore* (MIC) devices.

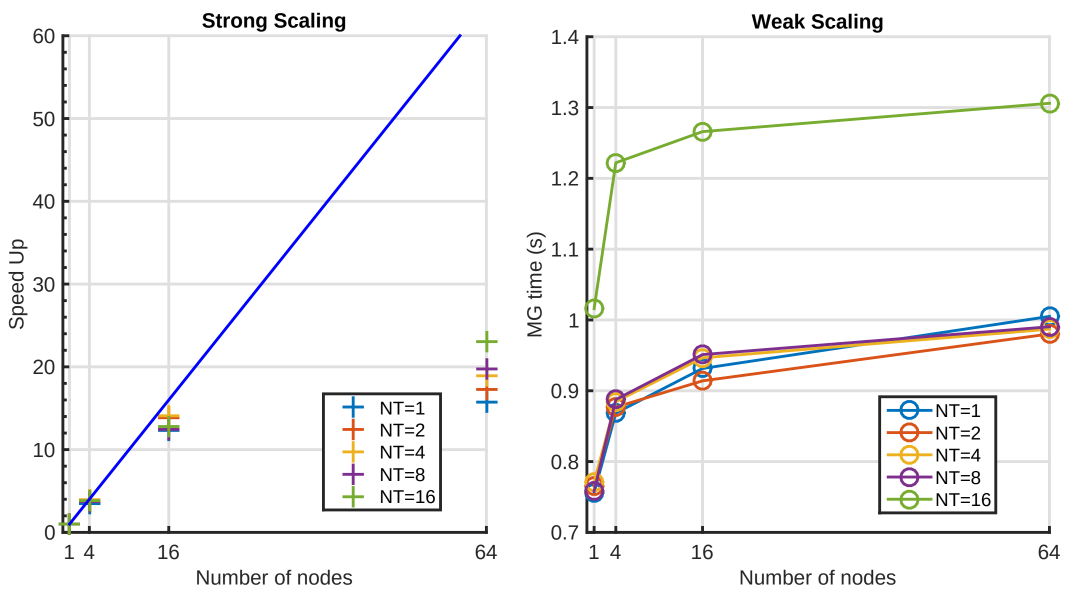


Figure 26: Strong scaling with a 1024×4096 grid size (left) and weak scaling (right) with grid sizes $1024 \times 4096, 10248 \times 8192, 4096 \times 16384$ and 8192×32768 respectively for 1, 4, 16 and 64 nodes.

References

[1] W.L. Briggs, V.E. Henson and S.F. McCormick, A Multigrid Tutorial, Second Edition, SIAM (2000).

[2] `http://graal.ens-lyon.fr/MUMPS/`.

[3] `Multigrid Formulation for Finite Elements`,
`https://crppsvn.epfl.ch/repos/bsplines/trunk/multigrid/docs/multigrid.pdf`

[4] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition , SIAM, (1994).

[5] P. Wesseling, An Introduction to Multigrid Methods, Edwards, 2004.

[6] X. Lapillonne, S. Brunner, T. Dannert, S. Jolliet, A. Marinoni et al., Phys. Plasmas 16, 032308 (2009).