Parallel GMRES with Polynomial Preconditioning

E. Stenhede[*], Z.N. Meng[*], J.A. Sanders[*]

[*]Special Topics in Computational Science and Engineering, TU Delft

Introduction

The Generalized Minimal Residual (GMRES) method is an iterative technique widely employed for solving large sparse linear systems. We have utilized it to address a 3D convection-diffusion problem

$$\begin{aligned} -\Delta u + \frac{\partial u}{\partial x} &= f, & &(x,y,z) \in [0,1]^3, \\ u(x,y,0) &= g(x,y), & &(x,y) \in [0,1]^2, \\ u(x,y,z) &= 0, & &\text{elsewhere on boundary,} \end{aligned}$$

[1] [2]

We have employed a finite difference method stencil and a parallel GMRES solver using MPI. To increase the convergence speed, a polynomial preconditioner in conjunction with diagonal scaling was used.

Polynomial Preconditioners for GMRES

RQ1: How does the increased computational intensity of Neumann preconditioning weigh against the need for less total iterations?

After applying diagonal scaling to the system, the truncated Neumann series $P_n(A)$ can be used as a polynomial preconditioner. It is defined as

$$P_n(A) = \sum_{k=0}^{n} (I - A)^k \approx A^{-1}. \tag{1}$$

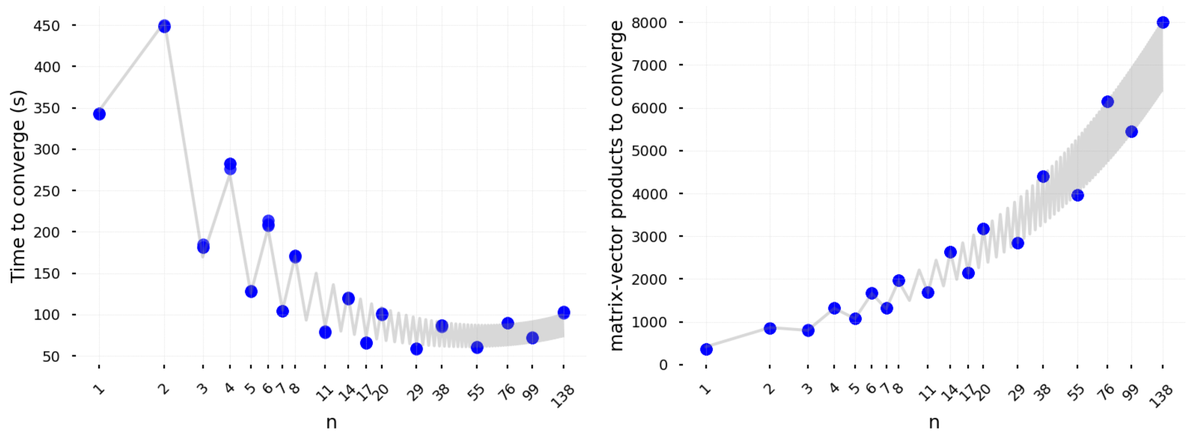

Figure 1: Comparison of GMRES solver with different polynomial preconditioner degrees. In this figure, 48 cores and a cartesian communication protocol are used to solve a system with grid size $n = 300 \times 300 \times 300$.

Even though the number of matrix-vector products are increasing with n, the total time to convergence is decreasing up to a point, because the number of calls to `dot` and `axpby` are decreased.

The oscillating behaviour seen in Figure 1 is caused by the terms in (1), the conclusion being that odd orders should always be used.

Ghost Node Communication Protocols

RQ2: How does MPI communication protocols scale for ghost node interchange in parallel stencil-based matrix-vector products

Three methods of MPI communication were benchmarked, namely `Isend/Irecv` using a global communication protocol (`MPI_COMM_WORLD`), `Isend/Irecv` using a cartesian communication protocol, and one sided communication using `MPI_Win` and `MPI_Get`.

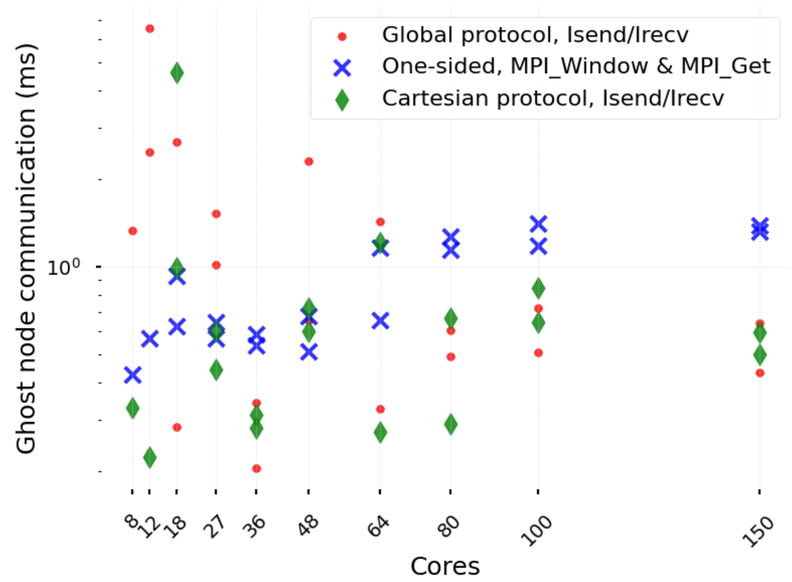

Figure 2: Mean time spent communicating ghost nodes during stencil-vector application. One-sided communication scales poorly, when using more than one node (more than 48 threads). When communicating within one node, the results are inconclusive.

Drawing a definitive conclusion from data benchmarked utilizing a single compute node with 48 cores poses a challenge. When the program executes across multiple nodes, we find that the one-sided communication scheme exhibits higher time consumption in data communication compared to the other two schemes.

Dot Communication Time

RQ3: How does the data communication of dot product influence the computational time of GMRES as the number of compute nodes increases?

Performing scalar products on multiple processes necessitates invoking `MPI_Allreduce` to combine values from all processes and distribute the result back to all processes. Figure 3 shows that the reduction overhead grows rapidly as the number of nodes reaches 8. When an adequate number of compute nodes are available, the Cartesian communication of `MPI_Allreduce` does not significantly impact the computational time.

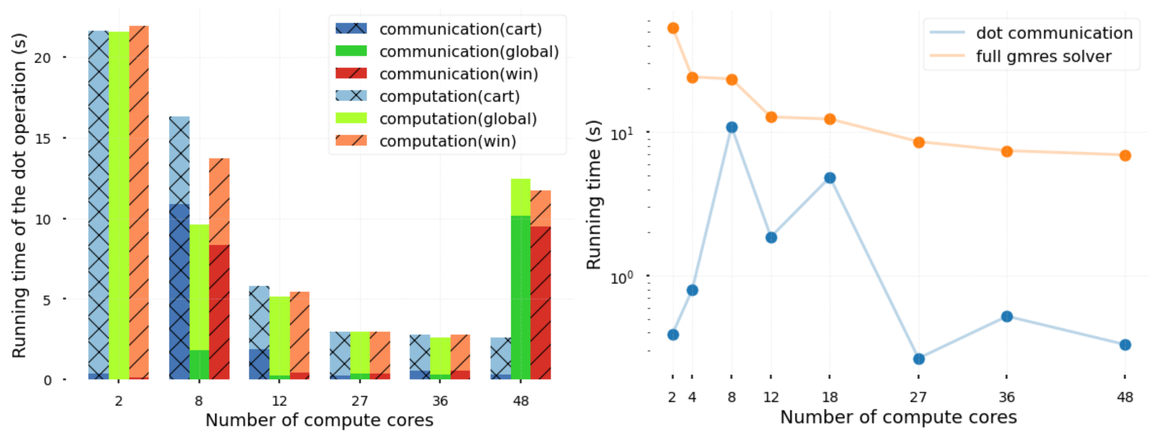

Figure 3: Time spent on computation and communication for each communication scheme in the scalar product operation (left) and comparison between the reduction overhead and the running time of the entire solver with the cartesian communicator (right). In this figure, GMRES iterates 50 times on a $300 \times 300 \times 300$ grid.

However, There is an unexpectedly large overhead in the global and one-sided communication scheme when there are 48 compute cores. We observe that the Cartesian communicator can optimize the data traffic within NUMA.

[1] $f(x,y,x) = z\sin(2\pi x)\sin(\pi y) + 8z^3$
[2] $g(x,y) = x(1-x)y(1-y)$