EasyChair Preprint

№ 6936

Scalable Hybrid Parallel ILU Preconditioner to Solve Sparse Linear Systems

Raju Ram, Daniel Grünewald and Nicolas R Gauger

October 26, 2021

Scalable hybrid parallel ILU preconditioner to solve sparse linear systems

Raju Ram[1,2], Daniel Grünewald[1], and Nicolas R Gauger[2]

[1] Fraunhofer ITWM, Competence Center High Performance Computing, Kaiserslautern, Germany
[2] Chair for Scientific Computing, Technische Universität Kaiserslautern, Germany

Abstract. Incomplete LU(ILU) preconditioners are widely used to improve the convergence of general-purpose large sparse linear systems in computational simulations because of their robustness, accuracy, and usability as a black-box preconditioner. However, the ILU factorization and the subsequent triangular solve are sequential for sparse matrices in their original form. Multilevel nested dissection (MLND) ordering can resolve that issue and expose some parallelism. This work investigates the parallel efficiency of a hybrid parallel ILU preconditioner that combines a restricted additive Schwarz (RAS) method on the process level with a shared memory parallel MLND Crout ILU method on the core level. We employ the GASPI programming model to efficiently implement the data exchange on the process level. We show the scalability results of our approach for the convection-diffusion problem.

Keywords: Sparse linear systems · Parallel ILU preconditioner · Domain decomposition · GASPI · METIS · Task-level parallelism

1 Research problem

Solution of the large sparse linear system $Ax = b$, arising after discretization of the partial differential equations (PDE), is one major computational task of chemistry, physics, and engineering-based simulations. Krylov subspace-based iterative methods are preferred over direct methods due to lesser time complexity and memory requirements. These solvers use preconditioners to accelerate their convergence. Incomplete LU (ILU) is widely used as a preconditioner because of its robustness, accuracy, and usability as a black-box preconditioner for general purpose (asymmetric, indefinite) linear systems. On the other hand, parallel Krylov solvers with good scalability features are required to fully exploit the increasing parallelism provided by modern hardware. Scalability measures the parallel efficiency of implementation. Better scalability allows simulation of more detailed models, more precise parameter studies, and more cost-efficient resource utilization. The scalability of ILU-based Krylov solvers is restricted due to the sequential nature of preconditioner operations such as factorization and solution to the triangular systems. We combine the thread-level parallelism approach described in [1] with Schwarz preconditioners at the distributed level to address the scalability challenges in ILU preconditioner on modern hardware.

2 Methodology

We propose a two-level domain decomposition (DD) preconditioner following a hybrid execution model which fits the memory hierarchies of modern hardware architectures well. For distributed memory parallelism, we use the GASPI communication API [3] since it provides fine-grained communication across processes. The communication is single-sided, asynchronous, and is complemented by lightweight synchronization primitives. For shared-memory parallelism, we use data dependency-driven task-based parallelism using pthreads.

2.1 Distributed memory parallelism

We use the Additive Schwarz (AS) method at the first level of DD and associate one sub-domain with each GASPI process. Thereby, the vertex set V of the graph corresponding to the matrix A is decomposed into N non-overlapping sub-domains V_i^0 such that $V = \bigcup_{i=1}^{N} V_i^0$ and $V_i^0 \bigcap V_j^0 = \emptyset$ for $i \neq j$. This decomposition may be augmented by a so called δ-overlap to generate partitions V_i^δ ($\delta \geq 1$), where $V_i^\delta \supset V_i^0$ is obtained by including all the immediate neighboring vertices of the vertices in V_i^0 up to distance δ. Restriction operators $R_i^\delta \in \mathbf{R}^{|V_i^\delta| \times |V|}$ and scaling operators $D_i^\delta \in \mathbf{R}^{|V_i^\delta| \times |V_i^\delta|}$ associated with each V_i^δ and can be defined such that a partition of unity $\mathbb{1} = \sum_{i=0}^{N} (R_i^\delta)^T D_i^\delta R_i^\delta$ is formed. Here, the transpose $(R_i^\delta)^T$ corresponds to the expansion operator. Then, AS decomposes the global problem $Ax = b$ into sub-domain solve problems $A_i x_i = b_i$, which can be solved in parallel and whose solutions are patched together a posteriori. The sub-domain matrix A_i is defined as $A_i := (R_i^\delta A (R_i^\delta)^T)$. Depending on the sub-domain partitioning, different preconditioners can be implemented:

1) Non-overlapping AS preconditioner : $M_{AS}^{-1} = \sum_{i=1}^{N} (R_i^0)^T A_i^{-1} R_i^0$

2) Restricted AS (RAS) preconditioner: $M_{RAS}^{-1} = \sum_{i=1}^{N} (R_i^0)^T D_i^\delta A_i^{-1} R_i^\delta$.

We use $\delta = 1$ in RAS which is known to converge faster than AS method [2].

2.2 Shared memory parallelism

The global matrix A loses coupling information across sub-domains in the first level of DD approach. This effect becomes more severe with increasing number of sub-domains. To prevent this, we introduce the second level of DD that partitions the distributed memory subdomain further using multilevel nested dissection(MLND) as described in [1]. MLND preserves the information of the matrix A_i and allows to obtain fine granular parallelism. We use the multi-threading version of METIS to generate the MLND permutation Π in our implementation. MLND reorders A_i into $A_{i,perm}$ such that $A_{i,perm} = \Pi^T A_i \Pi$. Independent local matrices are extracted from $A_{i,perm}$ which are then factorized in a task-parallel way. Similarly, we solve the triangular system using the same MLND task tree structure exploiting the local dependency in the tasks. We provide a custom implementation for performance critical sparse vector, used during serial Crout ILU factorization. Our sparse vector implementation is significantly faster than C++ STL based data structures such as std::map and std::unordered_map [4].

3 Preliminary results

We discretize the following 3D convection-diffusion PDE using second order finite differences on a regular rectangular mesh in an unit cube $(x, y, z) \in \Omega = (0, 1)^3$.

$$\Delta u + k^2 * x^2(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z}) = f(x, y, z),\ k^2 = 100 \tag{1}$$

We set $f(x, y, z)$ such that the solution $u(x, y, z)$ of the above PDE is equal to $exp(xyz) * sin(\pi x) * sin(\pi y) * sin(\pi z)$ and use Dirichlet boundary conditions as $u(\partial\Omega) = f(\partial\Omega)$. We solve the linear system using GMRES(30) solver with termination criteria of relative residual as 10^{-9}. The performance is evaluated on a cluster of 2.4 GHz Intel(R) Xeon(R) Gold 6148 CPU dual socket nodes, each socket with 20 cores which are connected by EDR Infiniband interconnects. We start one GASPI process per socket in our experiments.

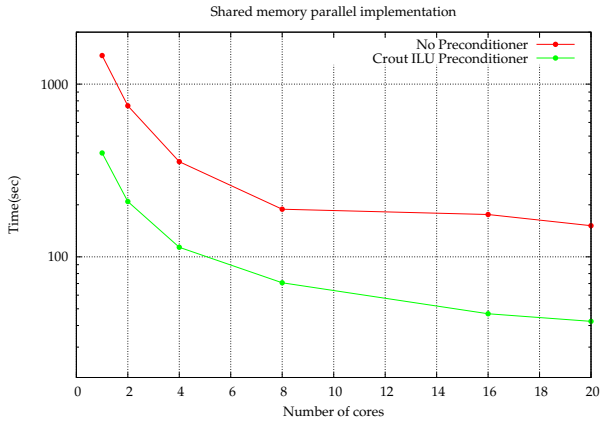

Fig. 1: GMRES runtime(s) for matrix size 8 million and MLND tree height as 5

First, we investigate the shared memory performance. On 1 GASPI process with 20 cores, MLND Crout ILU preconditioned GMRES achieves the parallel efficiency of **47.21**%. This allows to obtain **3.58**x gain in preconditioned GMRES run-time compared to plain GMRES on 20 cores (fig. 1).

Second, we evaluate the performance of hybrid implementation. On 64 GASPI processes each having 20 cores, RAS preconditioned GMRES(30) achieves a parallel efficiency of **83.61**%. This is superior to the AS preconditioned GMRES(30) parallel efficiency of **65.60**% and is because RAS has limited the increase of GMRES(30) iterations for higher number of GASPI processes (table 1). We obtain **3.88**x gain in GMRES(30) run-time using MLND Crout ILU based RAS preconditioner in comparison to no preconditioner on 64 processes each having 20 cores (fig. 2).

Table 1: GMRES iterations with different Schwarz preconditioners

# GASPI processes	# Iterations using AS	# Iterations using RAS
1	326	326
8	390	333
16	452	348
32	539	378
64	724	425

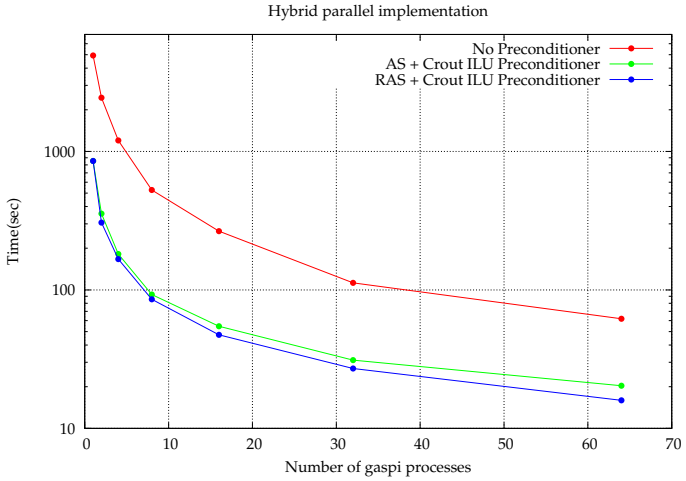

Fig. 2: GMRES runtime(s) for matrix size 64 million and 20 cores per process

4 Future outlook

The parallel efficiency of MLND Crout ILU based RAS preconditioner is promising. To handle real-world large problems, we have introduced more robust features such as row and col-based permutation, inverse-based droppings, and MC64 matching and currently testing them on various real-world linear systems.

References

1. Aliaga, J.I., Bollhöfer, M., Martı, A.F., Quintana-Ortı, E.S., et al.: Exploiting thread-level parallelism in the iterative solution of sparse linear systems. Parallel Computing **37**(3), 183–202 (2011)
2. Efstathiou, E., Gander, M.J.: Why restricted additive schwarz converges faster than additive schwarz. BIT Numerical Mathematics **43**(5), 945–959 (2003)
3. Grünewald, D., Simmendinger, C.: The GASPI API specification and its implementation GPI 2.0. In: Proceedings of the 7th International Conference on PGAS Programming Models. vol. 243 (2013)
4. Ram, R., Grünewald, D., Gauger, N.R.: Data structures to implement the Sparse Vector in Crout ILU preconditioner (2019), Sparse Days 2019