

Improvements of a Fast Parallel Poisson Solver on Irregular Domains

Andreas Adelmann¹, Peter Arbenz^{2*}, and Yves Ineichen^{1,2}

¹ Paul Scherrer Institut, Villigen, Switzerland

² ETH Zürich, Chair of Computational Science, Zürich, Switzerland

Abstract. We discuss the scalable parallel solution of the Poisson equation on irregularly shaped domains discretized by finite differences. The symmetric positive definite system is solved by the preconditioned conjugate gradient algorithm with smoothed aggregation (SA) based algebraic multigrid (AMG) preconditioning. We investigate variants of the implementation of SA-AMG that lead to considerable improvements in the execution times. The improvements are due to a better data partitioning and the iterative solution of the coarsest level system in AMG. We demonstrate good scalability of the solver on a distributed memory parallel computer with up to 2048 processors.

Keywords: Poisson equation, finite differences, preconditioned conjugate gradient algorithm, algebraic multigrid, data partitioning.

1 Introduction

The solver described in this paper is part of the general accelerator modeling tool Object Oriented Parallel Accelerator Library (OPAL) [3]. OPAL enables the solution of the most challenging problems in the field of high precision particle accelerator modeling. These include the simulation of high power hadron accelerators and of next generation light sources.

In these simulations the evolution of the charged particles is determined by the collisionless *Vlasov equation*. The most compute intense portion of the simulation is the determination of the electrostatic potential ϕ from the *Poisson equation*

$$-\varepsilon_0 \Delta\phi(\mathbf{x}) = \rho(\mathbf{x}), \quad (1)$$

in a coordinate system moving with the particles. Here, ρ denotes the spatial charge density and ε_0 is the dielectric constant. The electric field is obtained from

$$\mathbf{E} = -\nabla\phi. \quad (2)$$

An appropriate *Lorentz transformation* yields the electromagnetic fields in the static reference frame that are needed to move the particles. For details see [2].

*Corresponding author. Email: arbenz@inf.ethz.ch

The Poisson problem (1) discretized by finite differences can efficiently be solved on a rectangular grid by a Particle-In-Cell (PIC) approach [17]. The right hand side in (1) is discretized by sampling the particles at the grid points. In (2), ϕ is interpolated at the particle positions from its values at the grid points. We also note that the common FFT-based Poisson solvers and similar approaches [16, 17] are restricted to box-shaped or open domains.

In section 2 we present our finite difference approach and how we treat the boundary at curved surfaces. In section 3 we review the iterative system solver. In section 4 we discuss numerical experiments on 512–2048 processors of a Cray XT-5. In particular, we are interested in the partitioning of the computational domain and in the coarse level solver of the multilevel preconditioner. Section 5 concludes the paper.

2 The discretization

In this section we discuss the solution of the Poisson equation in a domain $\Omega \subset \mathbb{R}^3$ as indicated in Fig. 1. The boundary of the domain is composed of

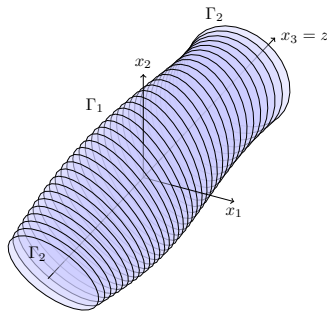


Fig. 1. Sketch of a typical domain

two parts, a curved, smooth surface Γ_1 and two planar portions at $z = -d$ and $z = +d$ that form together Γ_2 . In physical terms Γ_1 forms the casing of the pipe, while Γ_2 is the open boundary at the inlet and outlet of the beam pipe, respectively. The centroid of the particle bunch is at the origin of the coordinate system. In practice the shape of Γ_1 can be quite complicated. Our code assumes that a ray that emanates perpendicularly from the z -axis crosses Γ_1 at most once.

The Poisson problem that we are going to solve is given by

$$\begin{aligned} -\epsilon_0 \Delta \phi &= \rho \text{ in } \Omega, \\ \phi &= g \equiv 0 \text{ on } \Gamma_1, \quad \partial_n \phi + (1/d)\phi = 0 \text{ on } \Gamma_2. \end{aligned} \tag{3}$$

The parameter d in the Robin boundary condition is half the extent of Ω in z -direction [15].

We discretize (3) by a standard second order finite difference scheme defined on a rectangular grid with grid spacing h_i in the i -th coordinate direction. It is natural to arrange the grid such that the two portions of Γ_2 lie in grid planes.

A lattice point is called an *interior* point if all its direct neighbors are in Ω . All other grid points are called *near-boundary* points. At interior points \mathbf{x} we approximate $\Delta u(\mathbf{x})$ by the well-known 7-point difference star

$$-\Delta_h u(\mathbf{x}) = \sum_{i=1}^3 \frac{-u(\mathbf{x}-h_i \mathbf{e}_i) + 2u(\mathbf{x}) - u(\mathbf{x}+h_i \mathbf{e}_i)}{h_i^2}. \quad (4)$$

At grid points near the boundary we have to take the boundary conditions in (3) into account by constant, linear, or quadratic extrapolation [2, 7, 14].

The finite difference discretization just described leads to a system of equations

$$A\mathbf{x} = \mathbf{b}, \quad (5)$$

where \mathbf{x} is the vector of unknown values of the potential and \mathbf{b} is the vector of the charge density interpolated at the grid points. The Poisson matrix A is an M -matrix irrespective of the boundary treatment [11]. Constant and linear extrapolation lead to a *symmetric* positive definite (spd) A while quadratic extrapolation yields a *nonsymmetric* but still positive definite Poisson matrix.

The boundary extrapolation can introduce large diagonal elements in A . In order to avoid numerical difficulties it is advisable to apply a symmetric scaling to the system (5).

3 The solution method

If A is spd the conjugate gradient (CG) algorithm [11, 13] solves (5) in a fast and memory efficient way. In the case of quadratic boundary extrapolation A is nonsymmetric, however only ‘mildly’. There are some deviations from symmetry only at some of the boundary points. In this situation the CG algorithm is still applicable, i.e., it does not break down. However, it loses the finite termination property and may behave more like steepest descent [10]. In our experiments we observed a convergence behavior that did not deviate from the one for spd matrices.

To improve the convergence behavior of the CG methods we precondition (5) by smoothed aggregation-based algebraic multigrid (SA-AMG) preconditioners [8, 19]. Aggregation-based AMG methods cluster the fine grid unknowns to aggregates as representation for the unknowns on the next coarser grid.

The multigrid preconditioner and iterative solver are implemented with the help of the Trilinos framework [12, 18]. Trilinos provides state-of-the-art tools for numerical computation in various packages. The AztecOO package, e.g., provides iterative solvers and ML [8] provides multilevel preconditioners. By means of ML, we created our smoothed aggregation-based AMG preconditioner. We use ML’s ‘decoupled’ aggregation strategy [19] which constructs aggregates consisting of cubes of $3 \times 3 \times 3$ vertices. This strategy may entail non-optimal aggregates

at the subdomain interfaces. The subdomains represent the portion of the problem dealt with by a processor or core. The partitioning in subdomains is done manually based on the encasing cubic grid.

As suggested in [1] we choose a Chebyshev polynomial smoother. The employed coarse level solver (Amesos-KLU) ships the coarse level problem to node 0 and solves it there by means of an LU factorization. An alternative is to apply a few steps of an iterative solver (e.g. Gauss–Seidel) at the coarsest level. A small number of iteration steps decreases the quality of the preconditioner and thus increases the PCG iteration count. A large number of iteration steps increases the time for applying the AMG preconditioner. In [2] we found three Gauss–Seidel iteration steps to be a good choice for our application. In this paper, we use Chebyshev iteration.

4 Numerical experiments

In the recent paper [2] we conducted numerical experiments on the Cray XT-4 at the Swiss National Supercomputing Centre (CSCS) in order to assess the performance and scalability of our solver. In particular, we compared our solver with an FFT-based one. This is in fact not completely trivial, since the computational domains of the two solvers differ. The FFT-based solver requires a rectangular computational domain. Usually Neumann (free) boundary conditions are applied. Our finite difference solver approximates well the geometry of the device and uses homogeneous Dirichlet boundary conditions. In situations where the boundary is far away from the particle bunch the solutions of the two problems match quite well. However, in problems where the spatial extent of the beam is comparable with that of the beam pipe it is important to have an accurate representation of the field near the boundary. Then, the results of the computations can differ significantly, and the results of the FFT-based solver are questionable. Nevertheless, the FFT-based solver can be faster than our iterative solver by up to about a factor 5.

In this paper we discuss the issue of load balancing and communication overhead. We conduct the numerical experiments on the newest Cray XT-5 at CSCS [5]. This machine consists of 3688 AMD hexa-core Opteron processors clocked at 2.4 GHz. The system has 28.8 TB of DDR2 RAM, 290 TB disk space, and a high-speed interconnect with a bandwidth of 9.6 GB/s and a latency of $5 \mu\text{s}^\ddagger$.

Our computational domains are embedded in a 3D rectangular domain, as illustrated in Fig. 1. The rectangular computational grid is generated inside the rectangular domain. Only grid points of the rectangular grid that are included in the computational domain Ω are used in the computation. In the computations in [2] the partitioning of the computational domain was based on the partitioning of the *whole* rectangular domain. (This underlying rectangular grid is induced by the particle code OPAL [3] that participates at the overall computation.)

[‡]<http://www.cscs.ch/455.0.html> retrieved on July 13, 2010.

Therefore, some subdomains contained far fewer grid points than others, causing severe load imbalance. In fact, in many cases there were subdomains that contained no grid points at all. In our new approach we partition the computational domain to enhance load balancing. In this paper we compare old and new approach.

The computational domain we are dealing with in this paper is a circular cylinder embedded in a $1024 \times 1024 \times 1024$ grid. In this setup the problem size is still reasonably large when employing 2048 cores, i.e., a subcube contains (up to) $524'288$ grid points. We use linear extrapolation at the Dirichlet boundary Γ_1 . The solver is the AMG-preconditioned conjugate gradient algorithm as implemented in Trilinos and discussed in section 3. Timings for three phases of the computation are given in Table 1. For this large problem with 840 million degrees

cores	solution	construction	application	total ML	iterations
512	62.22 [1.00]	35.12 [1.00]	51.68 [1.00]	86.79 [1.00]	20
1024	32.95 [0.94]	19.95 [0.88]	27.47 [0.94]	47.41 [0.92]	20
2048	17.68 [0.87]	12.37 [0.71]	14.85 [0.87]	27.22 [0.80]	20

Table 1. Times in seconds and relative parallel efficiencies. The original data distribution is used, and the coarsest AMG level is solved with KLU.

of freedom we observe quite good efficiencies. The solver runs at 87% efficiency with 2048 cores relative to the 512-cores performance. Note that the solution phase contains the application of the preconditioner. Therefore, the difference of the two columns indicated by ‘solution’ and ‘application’ essentially gives the time for matrix-vector and inner products in the conjugate gradient algorithm. The column ‘total ML’ comprises the sum of columns ‘construction’ and ‘application’. The construction phase is performing the worst with an efficiency of 71%. We found that much of the time in the construction of the preconditioner goes into the factorization of the coarsest level matrix. We therefore decided to replace the direct solver KLU by an iterative procedure. We apply one step of the Chebyshev semi-iterative method [9] with polynomial degree 10. The tim-

cores	solution	construction	application	total ML	iterations
512	63.12 [1.00]	32.09 [1.00]	52.73 [1.00]	84.80 [1.00]	20
1024	33.54 [0.94]	16.31 [0.98]	28.04 [0.94]	44.35 [0.96]	20
2048	18.56 [0.85]	8.10 [0.99]	15.66 [0.84]	23.76 [0.89]	21

Table 2. Times in seconds and relative parallel efficiencies. The original data distribution is used, and the coarsest AMG level is solved iteratively.

ings for this approach are given in Table 2. The times for the construction of the preconditioner have been reduced considerably, at the expense of a slightly more

expensive solution phase. Now the construction phase scales perfectly. Notice that the iteration counts change only marginally.

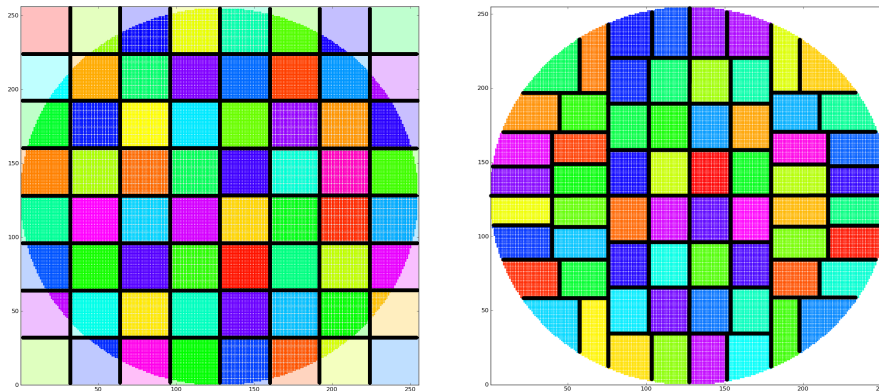


Fig. 2. (LEFT) Cross section of data distribution on 512 cores on a $8 \times 8 \times 8$ processor grid (colors indicate data owned by a processor) and (RIGHT) the same data redistributed with recursive coordinate bisection (RCB).

In Fig. 2 on the left a cross section is shown of the case where the computational domain is embedded in a cube that has been subdivided in $8 \times 8 \times 8$ subcubes. It is easily seen that the four subcubes in the corners do not contain any points of the computational grid, while other subcubes contain at least a few grid points. In general, a fraction of only about $\pi/4 \approx 0.8$ of the grid points of the cube will be included in the computational domain. This number corresponds to the ratio of the volumes of the circular cylinder and its encasing cube. That is, about 20% of the nodes in the cube are not involved in the computation and, hence about 20% of the subcubes contain a reduced number of nodes. Since subcubes correspond to cores, empty or almost empty subcubes correspond to idle or underloaded cores.

Evidently, this partitioning entails a severe load imbalance. Nevertheless, the speedups shown in Table 1 look good! These good speedups are comprehensible if one considers the most loaded processes that correspond to the innermost cubes, i.e., those close to the z -axis. These subcubes contain $1024^3/p$ nodes. An increase of the processor number p leads to an optimal speedup, at least as long as the floating point operations dominate the work load. This is the case here, as even in the 2048 processor computation a core handles more than half a million nodes.

Although speedups are good with this crude distribution of work, a better balanced work will lead to improved execution times. After all, there are only about 80% of the 1.1 billion nodes inside the computational domain.

To better balance the load we partition data using the recursive coordinate bisection (RCB) algorithm as it is implemented in the Zoltan toolkit [4, 6]. The

Trilinos package Isorropia[§] provides a matrix-based interface to Zoltan. The recursive coordinate bisection (RCB) algorithm partitions the graph according to the coordinates of its vertices. The computational domain is first divided into two subdomains by a cutting plane orthogonal to one of the coordinate axes so that half the work load is in each of the subdomains. (Notice that other fractions than 50-50 are possible and in fact needed if the number of processors is not a power of 2.) That coordinate axis is chosen which is associated with the longest elongation of the domain. Clearly, this procedure can be applied recursively. RCB leads to nicely balanced data distributions. In rectangular grids, RCB is a particularly fast partitioner since the coordinates of the grid vertices are easily determined from their indices. In Fig. 2 on the right the cross section of the circular cylinder is shown with the partitioning into 64 subdomains. Now, all subdomains contain an almost equal number of nodes which leads to almost equal loads per processor. This subdivision evidently is more complicated than the one on the left of this figure. Except for subdomains in the center, i.e. close to the z axis, there are more than just six neighbors (in 3D). With the increased number of neighbors the number of messages that are to be sent in the communication steps increases. The communication volume does not change much. (Notice that Trilinos constructs the communication pattern transparent to the user.)

cores	solution	construction	application	total ML	iterations
512	50.32 [1.00]	27.37 [1.00]	44.00 [1.00]	71.37 [1.00]	20
1024	28.14 [0.89]	16.82 [0.81]	24.82 [0.89]	41.58 [0.86]	20
2048	15.26 [0.82]	15.81 [0.43]	13.47 [0.82]	29.28 [0.61]	19

Table 3. Times in seconds and relative parallel efficiencies. Data is distributed by RCB. The coarsest AMG level is solved with KLU.

cores	solution	construction	application	total ML	iterations
512	51.08 [1.00]	25.65 [1.00]	44.89 [1.00]	70.55 [1.00]	20
1024	27.38 [0.93]	12.96 [0.99]	24.51 [0.92]	37.07 [0.95]	20
2048	14.76 [0.87]	6.69 [0.96]	13.10 [0.86]	19.79 [0.89]	19

Table 4. Times in seconds and relative parallel efficiencies. Data is distributed by RCB. The coarsest AMG level is solved iteratively.

In Tables 3 and 4 the execution times of our code is given with the data redistributed by RCB. These numbers are to be compared with those in Tables 1 and 2, respectively, where the original data distribution was used. For 512 cores the execution times are significantly smaller with the RCB distribution, about 20%, as the previous discussion suggests. Tables 5 and 6 give more details. There,

[§]See <http://trilinos.sandia.gov/packages/isorropia/>

cores	solution	construction	application	total ML
512	1.00 [0.81]	1.00 [0.78]	1.00 [0.85]	1.00 [0.82]
1024	0.89 [0.76]	0.81 [0.69]	0.89 [0.80]	0.86 [0.75]
2048	0.82 [0.71]	0.43 [0.55]	0.82 [0.74]	0.61 [0.66]

Table 5. Parallel efficiencies of the RCB partitioned runs and relative parallel efficiencies of the runs with original data distribution. The coarsest AMG level is solved with KLU.

cores	solution	construction	application	total ML
512	1.00 [0.81]	1.00 [0.80]	1.00 [0.85]	1.00 [0.83]
1024	0.93 [0.76]	0.99 [0.79]	0.92 [0.80]	0.95 [0.80]
2048	0.87 [0.69]	0.96 [0.79]	0.86 [0.72]	0.89 [0.74]

Table 6. Parallel efficiencies of the RCB partitioned runs and relative parallel efficiencies of the runs with original data distribution. The coarsest AMG level is solved iteratively.

in brackets, the efficiencies of the runs with the original rectangular distribution are listed relative to the 512 processor run with the RCB distribution.

When using the iterative solver on the coarsest level, speedups and thus efficiencies are quite close, cf. Tables 2 and 4. However, there are significant differences when the coarsest level system is solved directly. In this case the efficiencies deteriorate more quickly with the RCB distribution than with the original distribution. A more detailed analysis (not reproduced here) reveals that the significant difference between original and RCB partitioning is caused by the LU factorization of the matrix of the coarsest level. In fact, the factorization takes 3.64sec on 1024 cores and 7.86sec on 2048. We do not see a reason for this drastic increase of factorization time since the problem sizes are quite close (2048 vs. 1865). The fact that the partitions with RCB have more neighbors does not seem to be a strong enough reason. In any case, larger differences would have to be visible in Tables 2 and 4.

In this paper we restricted ourselves to problems posed on grids of size $1024 \times 1024 \times 1024$. In [2], investigating the original solver, we also included $512 \times 512 \times 512$ and $256 \times 256 \times 256$ grids. On the former grid the solver scaled similarly as on the 1024^3 grid. On the latter grid scalability was somewhat poorer. By replacing the direct coarse level solver by an iterative coarse level solver we expect similar improvements in the parallel performance of our solver also for these smaller problem sizes. As in the computations of this note, scalability will not be affected much by an improved partitioning, however load balance and execution time.

5 Conclusions

We have presented and discussed improvements of a scalable Poisson solver suitable to handle domains with irregular boundaries as they arise, for example, in beam dynamics simulations. The solver employs the conjugate gradient algorithm preconditioned by smoothed aggregation-based AMG. The code exhibits excellent scalability up to 2048 processors with cylindrical pipes embedded in meshes with up to 1024^3 grid points. We have reduced the execution time by about 20% by redistributing the data using the recursive coordinate bisection (RCB) algorithm. This removes the severe load imbalance of the original approach in [2]. Scalability was further improved by iteratively solving the linear system of equations on the coarsest level of the AMG preconditioner.

Acknowledgment

These computations have been performed in the framework of a Large User Project grant of the Swiss National Supercomputing Centre (CSCS). We acknowledge the help of the Cray support team at CSCS.

References

- [1] M. Adams, M. Brezina, J. Hu, and R. Tuminaro. Parallel multigrid smoothing: polynomial versus Gauss–Seidel. *J. Comput. Phys.*, 188(2):593–610, 2003.
- [2] A. Adelmann, P. Arbenz, and Y. Ineichen. A fast parallel Poisson solver on irregular domains applied to beam dynamics simulations. *J. Comput. Phys.*, 229(12):4554–4566, 2010.
- [3] A. Adelmann, C. Kraus, Y. Ineichen, and J. J. Yang. The Object Oriented Parallel Accelerator Library Framework. Technical Report PSI-PR-08-02, Paul Scherrer Institut, 2008-2010. http://amas.web.psi.ch/docs/opal/opal_user_guide-1.1.6.pdf.
- [4] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, C. Vaughan, Ü. Çatalyürek, D. Bozdog, W. Mitchell, and J. Teresco. *Zoltan 3.0: Parallel Partitioning, Load-balancing, and Data Management Services; User’s Guide*. Sandia National Laboratories, Albuquerque, NM, 2007. Tech. Report SAND2007-4748W http://www.cs.sandia.gov/Zoltan/ug_html/ug.html.
- [5] Cray Inc., Seattle, WA. *Cray XT5 Brochure*, 2009. Available from <http://www.cray.com/Products/XT/Systems/XT5.aspx>. Retrieved on July 13, 2010.
- [6] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Comput. Sci. Eng.*, 4(2):90–97, 2002.
- [7] G. E. Forsythe and W. R. Wasow. *Finite-difference methods for partial differential equations*. Wiley, New York, 1960.
- [8] M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, and M. G. Sala. ML 5.0 smoothed aggregation user’s guide. Tech. Report SAND2006-2649, Sandia National Laboratories, May 2006.
- [9] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

- [10] A. Greenbaum. *Iterative Methods for Solving Linear Systems*. SIAM, Philadelphia, PA, 1997.
- [11] W. Hackbusch. *Iterative solution of large sparse systems of equations*. Springer, Berlin, 1994.
- [12] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.
- [13] M. R. Hestenes and E. Stiefel. Methods of conjugent gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.
- [14] P. McCorquodale, P. Colella, D. P. Grote, and J.-L. Vay. A node-centered local refinement algorithm for Poisson’s equation in complex geometries. *J. Comput. Phys.*, 201(1):34–60, 2004.
- [15] G. Pöplau and U. van Rienen. A self-adaptive multigrid technique for 3-D space charge calculations. *IEEE Trans. Magn.*, 44(6):1242–1245, 2008.
- [16] J. Qiang and R. L. Gluckstern. Three-dimensional Poisson solver for a charged beam with large aspect ratio in a conducting pipe. *Comput. Phys. Commun.*, 160(2):120–128, 2004.
- [17] J. Qiang and R. D. Ryne. Parallel 3D Poisson solver for a charged beam in a conducting pipe. *Comput. Phys. Commun.*, 138(1):18–28, 2001.
- [18] The Trilinos Project Home Page. <http://trilinos.sandia.gov>.
- [19] R. S. Tuminaro and C. Tong. Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines. In *ACM/IEEE SC2000 Conference (SC2000)*, 2000. 21 pages, doi:10.1109/SC.2000.10008.