SpezialForschungsBereich F 32

Karl–Franzens Universität Graz
Technische Universität Graz
Medizinische Universität Graz

A Balanced Accumulation Scheme for Parallel PDE Solvers

M. Liebmann A. Neic G. Haase

SFB-Report No. 2013–004 March 2013

A–8010 GRAZ, HEINRICHSTRASSE 36, AUSTRIA

Supported by the
Austrian Science Fund (FWF)

Der Wissenschaftsfonds.

SFB sponsors:

- **Austrian Science Fund** (FWF)
- **University of Graz**
- **Graz University of Technology**
- **Medical University of Graz**
- **Government of Styria**
- **City of Graz**

A BALANCED ACCUMULATION SCHEME FOR PARALLEL PDE SOLVERS

MANFRED LIEBMANN†, AUREL NEIC†, AND GUNDOLF HAASE†

Abstract. We present a load balancing technique for a boundary data accumulation algorithm for non-overlapping domain decompositions. The technique is used to speed up a parallel conjugate gradient algorithm with an algebraic multigrid preconditioner to solve a potential problem on an unstructured tetrahedral finite element mesh. The optimized accumulation algorithm significantly improves the performance of the parallel solver and we show a nearly 50 percent runtime improvement over the standard approach in a benchmark run with 48 MPI processes. The load balancing problem itself is a global optimization problem that is solved approximately by local optimization algorithms in parallel that require no communication during the optimization process.

1. Introduction

A typical parallel implementation of the finite element method using the domain decomposition approach requires the exchange of data on the domain boundaries in order to implement parallel linear algebra operations as building blocks for Krylov-type algorithms [11]. Popular domain decomposition software packages like METIS [8] produce well balanced domain decompositions with respect to the number of finite elements per subdomain. This leads to well balanced parallel computation times, which are usually proportional to the number of finite elements in the subdomain. On the other hand the communication times are dependent on the number of boundary elements of the subdomains. Although METIS minimizes the global surface area of the boundaries, the boundary areas of the different subdomains can still vary significantly. This fact leads to unbalanced MPI communication times and thus to a loss in parallel efficiency in boundary data accumulation schemes. In this paper we present a parallel optimization scheme to globally reduce imbalances in the MPI processes during the boundary data accumulation by reassigning the responsibilities for the data accumulation on the boundary vertices in a balanced way. These imbalances furthermore counter code improvements based on detailed latency and bandwidth analysis, see [1] as a related work regarding algebraic multigrid. The reassignment introduces a unique master process for every boundary vertex that is shared between different subdomains and thus is shared between different MPI processes. The master process of a certain boundary vertex is responsible for the collection of the data values from other processes, and for the accumulation of these values, and the following redistribution of the accumulated value. Balancing the number of vertices for all master processes is thus the main goal of the redistribution process.

In Section 2 we present the problem statement and discuss the standard and optimized accumulation algorithm. In Section 3 we state the optimization problem

†Institute for Mathematics and Scientific Computing, University of Graz, Heinrichstrasse 36, 8010 Graz, Austria

E-mail addresses: manfred.liebmann@uni-graz.at, aurel.neic@uni-graz.at, gundolf.haase@uni-graz.at.

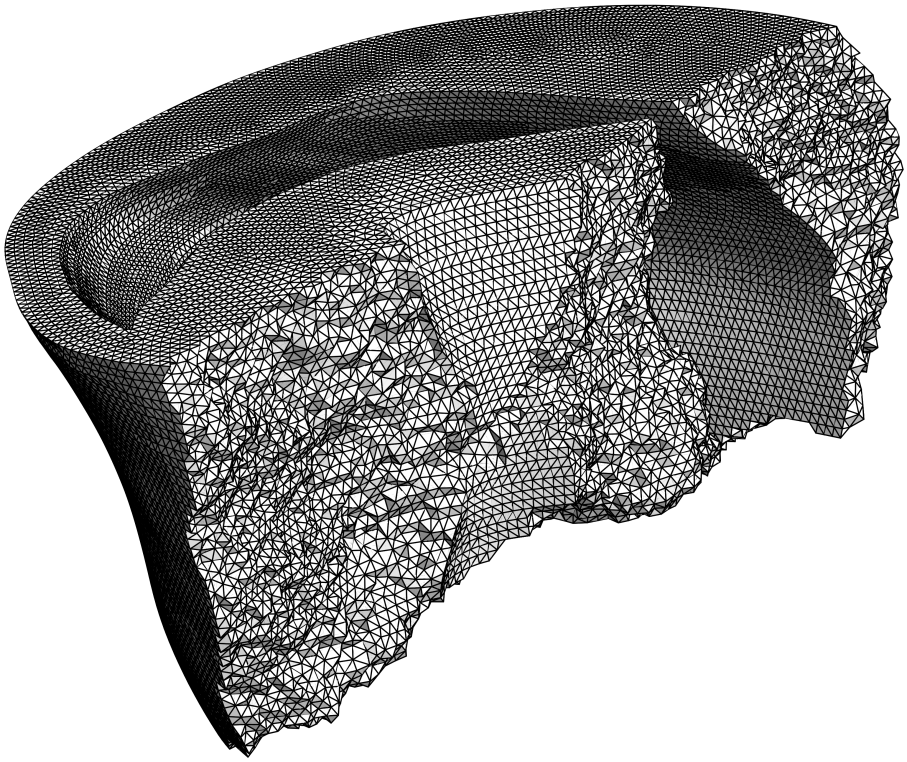

FIGURE 1. A discretized subdomain of the complex geometry used for the benchmark [16]

for balancing the optimized accumulation algorithm and discuss the parallel implementation of the algorithm. In Section 4 we introduce the benchmark problem and discuss benchmark results for the standard and the optimized accumulation algorithm in the context of a parallel conjugate gradient algorithm with algebraic multigrid preconditioner for an elliptic partial differential equation. Section 5 concludes with a discussion of the results.

2. Problem Statement

Let us consider a finite element simulation on a domain $\Omega \subset \mathbb{R}^3$ that is represented by a three-dimensional tetrahedral mesh. For parallel processing we partition the mesh into $P \in \mathbb{N}$ non-overlapping subdomains Ω_p, $p \in I := \{p \in \mathbb{N} : 1 \leq p \leq P\}$ using METIS [8].

$$\overline{\Omega} = \bigcup_{p=1}^{P} \overline{\Omega}_p \tag{1}$$

$$\Omega_p \cap \Omega_q = \emptyset, \quad p, q \in I,\ p \neq q \tag{2}$$

The subdomain boundaries are then naturally triangulated as shown in Figure 1 for our benchmark example and we define the boundary between two subdomains as

$$\Gamma_{p,q} := \overline{\Omega}_p \cap \overline{\Omega}_q, \quad p, q \in I,\ p \neq q\ . \tag{3}$$

We consider in this paper only non-overlapping finite element domain decompositions with uniquely distributed finite elements and duplicated subdomain boundary

vertices [6, 7]. Nevertheless, these results also can be immediately applied to other overlapping domain decompositions. In order to identify the set of vertices of the triangulation on the subdomain boundaries we first define the vertex index set $V_p \subset \mathbb{N}$ for the tetrahedral mesh that represents the subdomain $\overline{\Omega}_p$, $p \in I$. Then the index set of all vertices on $\overline{\Omega}$ is given by

$$V = \bigcup_{p=1}^{P} V_p \tag{4}$$

and the vertex indices for the boundary $\Gamma_{p,q}$ of two subdomains $\overline{\Omega}_p$ and $\overline{\Omega}_q$ can be calculated as the intersection

$$B_{p,q} := V_p \cap V_q, \quad p, q \in I,\ p \neq q\ . \tag{5}$$

The union of all boundary vertex indices of one subdomain $\overline{\Omega}_p$ is represented by the index set

$$B_p := \bigcup_{\substack{q=1 \\ q \neq p}}^{P} B_{p,q}, \quad p \in I \tag{6}$$

and the index set of all vertices on subdomain boundaries on the domain $\overline{\Omega}$ is the union

$$B = \bigcup_{p=1}^{P} B_p\ . \tag{7}$$

Under the assumption of linear finite element shape functions data has to be exchanged between subdomains for parallel computations only for the vertices on the subdomain boundaries [10]. Parallel finite element computations are typically set up using MPI [5] processes where a single MPI process handles the data and computations of a single subdomain. We assume for the simplicity of the presentation only scalar values to be associated with each vertex on the domain $\overline{\Omega}$.

The accumulation process that has to be typically performed for parallel linear algebra computations by all MPI processes involves the exchange of data values associated with vertices on the subdomain boundaries between MPI processes and the accumulation of these data values for each boundary vertex by each MPI process.

The standard accumulation (STD) is presented in Algorithm 1, whereas Algorithm 2 shows the optimized accumulation (OPT).

Algorithm 1 Standard Accumulation (STD)

MPI process p executes:
Require: $B_{p,q}$, f_p, $\tilde{f}_p$, $\tilde{g}_p$
CopyToBuffer($f_p, B_{p,q}, \tilde{f}_p$)
MPIAlltoall($\tilde{f}_p, B_{p,q}, \tilde{g}_p, B_{p,q}$)
AccumulateInPlace($\tilde{g}_p, B_{p,q}, f_p$)

Algorithm 2 Optimized Accumulation (OPT)

MPI process p executes:
Require: $S_{p,q}$, $R_{p,q}$, f_p, $\tilde{g}_p$
MPIAlltoall($f_p, S_{p,q}, \tilde{g}_p, R_{p,q}$)
Accumulate($\tilde{g}_p, R_{p,q}$)
MPIAlltoall($\tilde{g}_p, R_{p,q}, f_p, S_{p,q}$)

Algorithm 1, the standard accumulation algorithm, requires on the MPI process $p \in I$ all boundary vertex index sets $B_{p,q}$, $q \in I$ and the data vector f_p, where f_p holds the data values for all vertices in V_p. Furthermore the vectors $\tilde{f}_p$ and $\tilde{g}_p$ are temporary communication buffers. The tilde denotes the temporary nature of these vectors. The first function call CopyToBuffer fills the MPI send communication buffer $\tilde{f}_p$ according to the boundary vertex index distribution $B_{p,q}$. The second function call MPIAlltoall exchanges the the data values $\tilde{f}_p$ with other MPI processes and fills the MPI receive communication buffer $\tilde{g}_p$ with data values from other MPI processes. The third function call AccumulateInPlace directly adds all data values received in $\tilde{g}_p$ to the components of the vector f with the corresponding boundary vertex indices. Note that the data accumulation scheme can also be replaced by more sophisticated data handling schemes like interpolation or smoothing of boundary data [9], or the handling of Lagrange multipliers in Mortar methods [18] or FETI/BETI methods [15].

One advantage of the standard accumulation algorithm is the fact that only a single MPIAlltoall communication call is required. On the other hand there are serious disadvantages with the standard approach. One is that the CopyToBuffer function call creates an imbalance between the MPI processes due to the fact that the boundary vertices are not evenly distributed over all MPI processes. Furthermore the AccumulateInPlace function suffers from the same imbalance in the boundary vertex distribution when it executes the in-place-accumulation. This effect can be seen in Figure /reffig:cpuacc, which shows the CPU execution times in seconds of the standard (STD) and optimized (OPT) accumulation algorithm for 24 MPI processes for a typical benchmark run. As a further effect the collective MPIAlltoall communication call is also slowed down due to imbalances as shown in Figure 3. Overall the standard accumulation algorithm is slowed down by the CopyToBuffer, AccumulateInPlace, and the MPIAlltoall function calls. Actually filling the communication buffers takes significantly more time than the MPI communication over a modern QDR-Infiniband network. The overall timing of the standard accumulation algorithm is depicted in Figure 4.

Algorithm 2, the optimized accumulation algorithm, introduces several improvements over the standard approach. First of all the communication pattern is changed. Instead of exchanging data values with all processes that hold a specific boundary vertex the optimized approach introduces the concept of a master process for each boundary vertex. Only the master process is responsible for the accumulation of the data values of a specific boundary vertex and the communication with the other processes holding this specific boundary vertex. In the algorithm this is reflected by the two MPIAlltoall communication calls with asymmetric send and receive buffers. The send buffer configuration in the first MPIAlltoall call is denoted as $S_{p,q}$ and the receive buffer configuration is denoted as $R_{p,q}$. In the second MPIAlltoall call the roles of the send and receive buffers are exchanged. The Accumulate function accumulates the data values in-place within the buffer $\tilde{g}_p$. Thus the buffer acts as send and receive buffer for the MPIAlltoall communication functions.

To achieve a highly balanced MPI communication as shown in Figure 3 it is necessary to evenly distribute the boundary vertices to the master processes. This leads to a global optimization problem that is stated in the next chapter. The main advantage of the optimized accumulation algorithm is the one-to-one mapping of a boundary vertex to a master process. This means that now the MPIAlltoall communication function can directly use the data vector f_p as a send and receive buffer for the communication. Thus, no buffer copies are required and this is

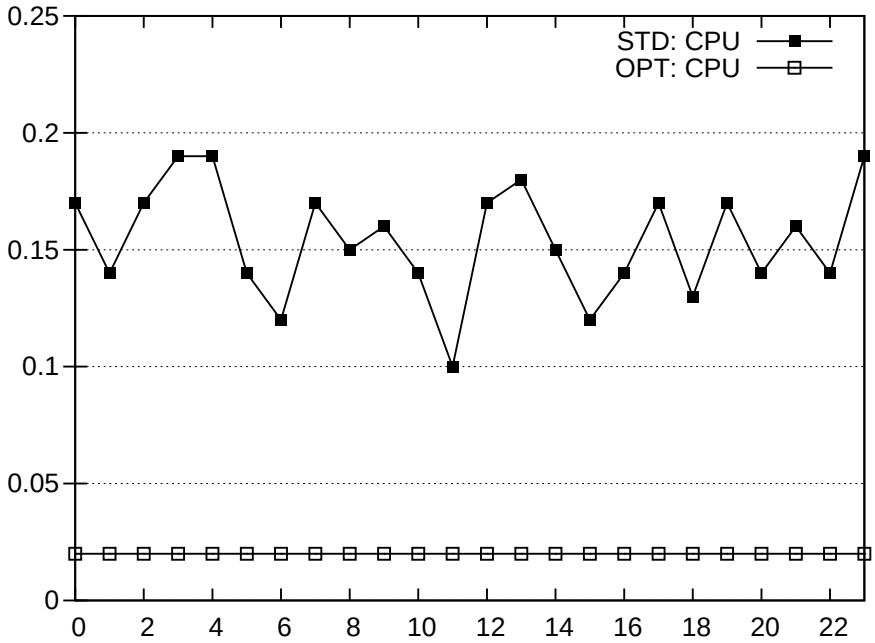

FIGURE 2. CPU execution times excluding MPI communication times in seconds for the standard (STD) and the optimized (OPT) accumulation algorithm for 24 MPI processes for the benchmark example measured with Scalasca.

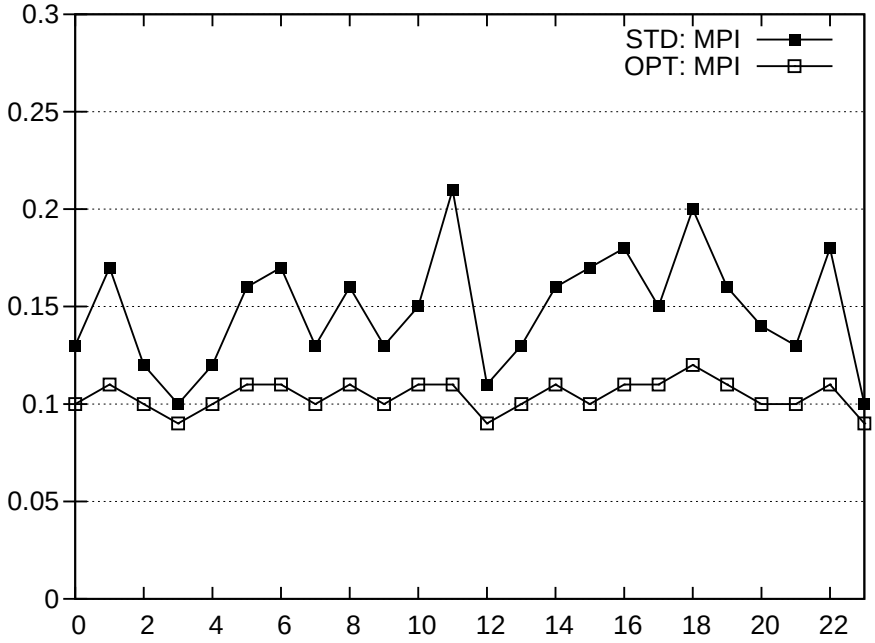

FIGURE 3. MPI communication times for the standard (STD) and the optimized (OPT) accumulation algorithm for 24 MPI processes for the benchmark example measured with Scalasca.

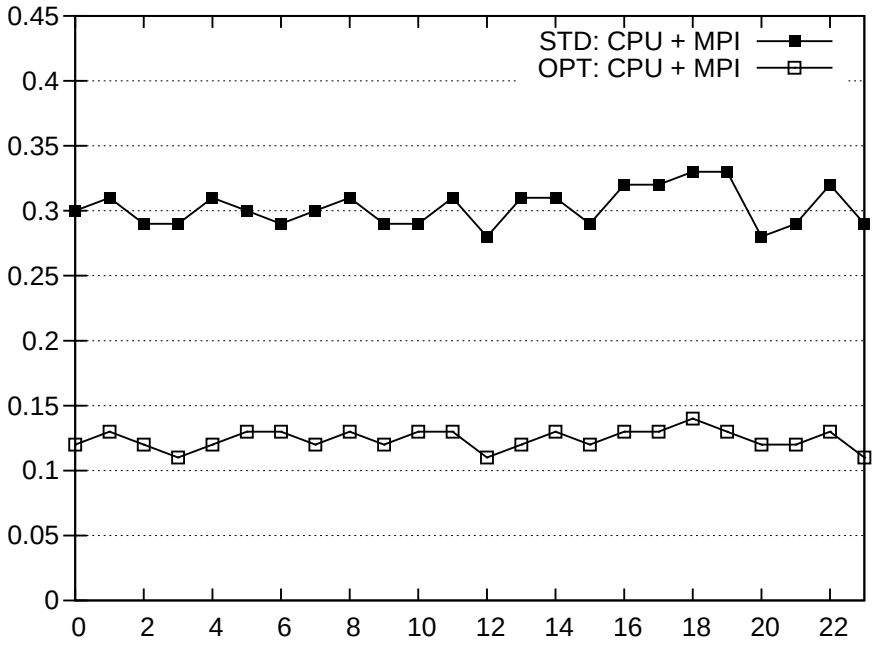

FIGURE 4. Total runtimes in seconds for the standard (STD) and the optimized (OPT) accumulation algorithm for 24 MPI processes for the benchmark example measured with Scalasca.

reflected in the huge relative CPU runtime advantage of the optimized accumulation algorithm shown in Figure 2.

3. Optimization Problem

To state the optimization problem we first have to define the set of pairs of connected boundary vertex indices and MPI process numbers.

$$E_p := \bigcup_{q=1}^{P} (B_{p,q} \times \{q\}), \quad p \in I \tag{8}$$

The set E_p is thus the edge set of the directed graph that connects the set of boundary vertex indices B_p on processor $p \in I$ to the set of MPI process numbers I. The adjacency matrix R^p of the directed graph is then defined by

$$R^p_{i,j} := \begin{cases} 1, & (i,j) \in E_p \\ 0, & \text{otherwise} \end{cases}, \quad i \in B_p,\ j \in I\ . \tag{9}$$

The goal of the global optimization algorithm is to assign in a balanced way a *master* MPI process $q \in I$ for every boundary vertex index in the global boundary index set, $v \in B$, under the constraint $(v,q) \in E_q$. To obtain an efficient parallel optimization algorithm we modify the optimization problem. First, every MPI process selects a set of boundary vertex indices $\hat{B}_p \subset B_p$ with a globally unique criterion.

$$\hat{B}_p := \{v \in B_p : p - 1 \equiv v - 1 \mod P\} \tag{10}$$

Note that the modulo criterion does not require any communication between the MPI processes for the vertex index selection to be unique. The corresponding

restriction of the edge set E_p is given by

$$\hat{E}_p := \left\{ (v,q) \in E_p : v \in \hat{B}_p \right\}, \quad p \in I \tag{11}$$

and the corresponding restriction of the adjacency matrix R^p is defined as

$$\hat{R}^p_{i,j} := \begin{cases} 1, & (i,j) \in \hat{E}_p \\ 0, & \text{otherwise} \end{cases}, \quad i \in \hat{B}_p,\ j \in I\ . \tag{12}$$

Second, a local optimization algorithm is executed on every MPI process $p \in I$ for the boundary vertex index set $\hat{B}_p$ again without communication. Third, after the local optimization algorithms have finished on all MPI processes the master MPI process distributions for the boundary vertex indices are exchanged between the MPI processes. This is the only step in the parallel optimization algorithm that requires communication.

To formulate the functional for the local optimization algorithm on MPI process $p \in I$ we have to define the target distribution $\bar{n}^p_q$ of the number of master MPI processes $q \in I$ assigned the boundary vertex index set $\hat{B}_p$. For this purpose we first define the number of boundary vertex indices N^p that fulfill the selection criterion as the cardinality of the boundary vertex index set $\hat{B}_p$.

$$N^p := \left| \hat{B}_p \right| \tag{13}$$

Note that the distribution of the boundary vertex indices is unique and with the global number of boundary vertex indices defined as $N := |B|$ we get the relation

$$N = \sum_{p=1}^{P} N^p\ . \tag{14}$$

We choose as target distribution for the number of master processes the approximate uniform distribution

$$\bar{n}^p_q := \left\lfloor \frac{\nu^p_q N^p + N^p}{P} \right\rfloor - \left\lfloor \frac{\nu^p_q N^p}{P} \right\rfloor, \quad p,q \in I \tag{15}$$

with the shift function defined as

$$\nu^p_q := q + p - 2 \mod P, \quad p,q \in I\ . \tag{16}$$

The purpose of the shift function is to distribute the remainder of the integer division N^p/P more evenly across all MPI processes. Note that for the uniform distribution holds

$$\bar{n}^p_q \in \left\{ \left\lfloor \frac{N^p}{P} \right\rfloor, \left\lfloor \frac{N^p}{P} \right\rfloor + 1 \right\}, \quad q,p \in I \tag{17}$$

and

$$N^p = \sum_{q=1}^{P} \bar{n}^p_q, \quad p \in I\ . \tag{18}$$

Although, more important is the sum over the distribution $\bar{n}^p_q$ with respect to all MPI processes $p \in I$.

$$N_q := \sum_{p=1}^{P} \bar{n}^p_q, \quad q \in I \tag{19}$$

Our main goal of the optimization process is to balance N_q, the number of boundary vertex indices a MPI process $q \in I$ is responsible for in the optimized accumulation algorithm. Since every MPI process locally optimizes towards an approximate uniform master distribution, the sum of these distributions is again an approximate uniform distribution, as desired. The implementation of the shift function avoids the accumulation of the remainders of the integer division N^p/P on a single MPI

process. Without the shift the last MPI process $p = P$ would always get a remainder contribution and thus would distort the desired uniform distribution. In any case the global optimization target is consistent due to the fact that

$$N = \sum_{q=1}^{P} N_q \; . \tag{20}$$

Define the master set $M^p \subset \hat{B}_p \times I$, $p \in I$ as all pairs of corresponding boundary vertex indices and master MPI process numbers. The master MPI process distribution for the master set M^p is then given by

$$n_q^p := \left| \left\{ v \in \hat{B}_p : (v, q) \in M^p \right\} \right|, \quad p, q \in I \; . \tag{21}$$

With these definitions we can finally define the local functional for the optimization problem.

$$J^p = \sum_{q=1}^{P} \left(n_q^p - \bar{n}_q^p \right)^2, \quad p \in I \tag{22}$$

The quadratic functional measures the deviation of the master distribution $n_q^p,\ q, p \in I$ from the approximate uniform distribution $\bar{n}_q^p,\ q, p \in I$. The global functional is defined as the sum over the local functionals.

$$J = \sum_{p=1}^{P} J^p \tag{23}$$

Algorithm 3 Optimization algorithm (SIM)

MPI process p executes:
Require: $\bar{n}^p$, r^p, c^p, d^p, K; l^p, n^p, m^p
  for $j = 1 \to N^p$ **do**
    $l_j^p \leftarrow (2^{31} - 1) * j \mod c_j^p$
    $b \leftarrow r^p_{d_j^p + l_j^p}$
    $m_j^p \leftarrow b$
    $n_b^p \leftarrow n_b^p + 1$
  end for
  $J^p \leftarrow \sum_{q=1}^{P} \left(n_q^p - \bar{n}_q^p \right)^2$
  $k \leftarrow 1$
  while $J^p \neq 0 \wedge k \leq K$ **do**
    for $j = 1 \to N^p$ **do**
      $l_j^p \leftarrow l_j^p + 1 \mod c_j^p$
      $a \leftarrow m_j^p$
      $b \leftarrow r^p_{d_j^p + l_j^p}$
      if $(n_a^p - \bar{n}_a^p) - (n_b^p - \bar{n}_b^p) \geq 1$ **then**
        $m_j^p \leftarrow b$
        $n_b^p \leftarrow n_b^p + 1$
        $n_a^p \leftarrow n_a^p - 1$
        $J^p \leftarrow J^p + 2\left[1 - (n_a^p - \bar{n}_a^p) + (n_b^p - \bar{n}_b^p)\right]$
      end if
    end for
    $k \leftarrow k + 1$
  end while
  return m^p, J^p

The local optimization algorithm (SIM) that executes on MPI process $p \in I$, shown in Algorithm 3, implements a randomized local search strategy inspired by the simulated annealing approach [17, 3]. The input parameters are the vector of the local approximate uniform distribution $\bar{n}^p \in \mathbb{N}^P$, the three vectors of the compressed row storage (CRS) representation of the restricted adjacency matrix $\hat{R}^p$, where $r^p \in I^{|\hat{E}_p|}$ is the vector of the column indices, $c^p \in \mathbb{N}^{N^p}$ is the vector of the number of non-zero elements per row, and $d^p \in \mathbb{N}^{N^p}$ is the vector of the one-based displacements to the beginning of the matrix rows in the vector r^p. The integer $K \in \mathbb{N}$ is the iteration limit. The temporary vector $l^p \in \mathbb{N}^{N^p}$ stores the current positions of the master MPI process number in the CRS data structure for the round robin update and the temporary vector $n^p \in \mathbb{N}^P$ stores the current master distribution. The output vector $m^p \in I^{N^p}$ is the master MPI process number distribution for the boundary vertex set $\hat{B}_p$ and the value of the functional J^p.

The algorithm starts with a pseudo-random initialization of the master distribution n^p and the calculation of the initial functional J^p. The algorithm then proceeds with local updates of the master distribution until the functional is zero, or a maximum number of iterations has been reached. In every iteration step a new master is selected from the set of master nodes for the given boundary vertex index $v \in \hat{B}_p$ in a round robin fashion. If the master MPI process number changes from a to b, then n_a^p will be decremented by one and n_b^p will be incremented by one and the functional changes according to

$$\begin{aligned} J^p_{\text{new}} - J^p_{\text{old}} &= (n^p_a - 1 - \bar{n}^p_a)^2 + (n^p_b + 1 - \bar{n}^p_b)^2 \\ &- (n^p_a - \bar{n}^p_a)^2 - (n^p_b - \bar{n}^p_b)^2 \\ &= 2\,(1 - (n^p_a - \bar{n}^p_a) + (n^p_b - \bar{n}^p_b))\ . \end{aligned} \tag{24}$$

To decide if the change will be accepted we use the criterion $J^p_{\text{new}} - J^p_{\text{old}} \leq 0$, or equivalently

$$(n^p_a - \bar{n}^p_a) - (n^p_b - \bar{n}^p_b) \geq 1\ . \tag{25}$$

This criterion allows changes that keep the functional constant and thus prevents the optimization process to get stuck prematurely in a local minimum.

4. Benchmark

To validate the performance of the standard and optimized accumulation algorithms we solve an elliptic PDE, discretized with linear tetrahedral finite elements, that describes the electric potential on the rabbit heart [13, 14] shown in Figure 5, with a parallel conjugate gradient algorithm with algebraic multigrid preconditioner (PCG-AMG). The parallel algebraic multigrid (AMG) solver uses a simple agglomeration based setup scheme and implements a V-cycle with a single Jacobi pre- and post-smoothing step [2, 16, 10].

The setup of the finite element simulation for the electric potential problem on cardiac tissue

$$-\nabla \cdot (\sigma\, \nabla u) = f, \quad \text{in } \Omega \tag{26}$$

$$u = g, \quad \text{on } \partial\Omega \tag{27}$$

was done with the cardiac arrhythmia research package (CARP) [12]. The tetrahedral mesh of the simulation domain Ω, which represents the geometry of a simplified model of a rabbit heart, contains 5,082,272 elements and 862,515 vertices. For more details on the original problem setting and its solution we refer the reader to [13, 14].

The benchmark runs were performed with 6, 12, 24, and 48 processor cores on the Mephisto cluster. The cluster contains five compute nodes with two six-core

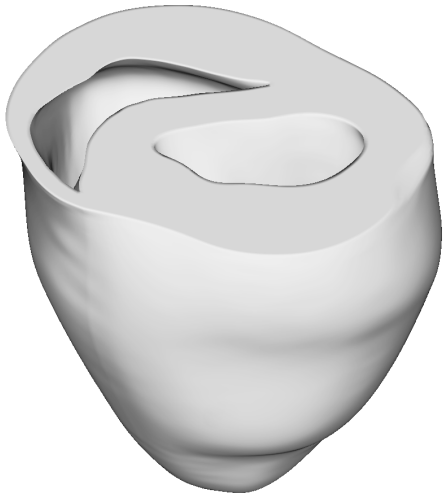

FIGURE 5. Image of the TBunnyC finite element mesh provided by G. Plank

TABLE 1. Total runtimes in seconds and parallel efficiency for the PCG-AMG solver for two linear solves.

	Runtime		Efficiency	
Processors	OPT	STD	OPT	STD
6	2.38	2.40	1.00	1.00
12	1.22	1.24	0.97	0.97
24	0.69	0.77	0.86	0.78
48	0.57	0.84	0.52	0.36

TABLE 2. MPI communication times in seconds for the standard (STD) and optimized (OPT) accumulation algorithms measured with Scalasca.

	OPT			STD		
Processors	min	mean	max	min	mean	max
6	0.03	0.05	0.06	0.02	0.07	0.08
12	0.04	0.05	0.06	0.03	0.06	0.14
24	0.09	0.11	0.12	0.10	0.15	0.21
48	0.27	0.27	0.28	0.52	0.53	0.58

Intel Xeon X5650 CPUs running at 2.67 GHz and 96 GB RAM per node and is connected with a Mellanox 40 Gb/s QDR-Infiniband network.

In our benchmarks we measured the total runtime of two consecutive PCG-AMG solver runs with a relative accuracy goal of 10^{-12} with a different number of processors and compute nodes. The total runtimes and parallel efficiency with respect to strong scaling are given in Table 1. The difference in the setup times for the PCG-AMG solver with respect to the standard and optimized accumulation algorithm turned out to be negligible. The setup of the parallel communication including the optimized accumulation algorithm is only about 10 percent slower

TABLE 3. Computed values of the functional J for different levels of the AMG algorithm and different numbers of processors.

	Processors			
Level	6	12	24	48
1	0	0	4	146
2	0	0	10	168
3	0	2	14	372
4	0	12	70	420
5	0	12	88	262
6	0	4	38	82
7	0	2	16	18
8	0	0	6	8
9	0	0	2	4
10	0		2	2

than the standard approach. While the parallel communication setup itself takes only about one percent of the total PCG-AMG setup time. Table 1 shows that the parallel efficiency of the PCG-AMG solver with the optimized accumulation algorithm is significantly improved. With 48 processors the performance advantage of the PCG-AMG solver with the optimized accumulation algorithm manifests in a speedup of 47 percent. Table 2 contains the minimum, maximum and average MPI communication time of both accumulation algorithms. The OPT algorithm outperforms STD with a significantly reduced average communication time as well as smaller deviations.

For the detailed performance analysis shown in the Figures 2, 3, 4 we used the parallel profiling software Scalasca [4, 19]. Although Scalasca provides a detailed view of many aspects of a parallel program, the instrumentation introduces a certain overhead that leads to increased runtimes. In this light the time measurements with Scalasca cannot be directly compared to the runtimes of optimized code.

Figure 2 shows the CPU execution times excluding the MPI communication times for the standard and optimized accumulation algorithms for 24 MPI processes. It is clearly visible that the standard accumulation algorithm suffers from the unbalanced boundary vertex distribution with huge fluctuations in the CPU runtimes across all MPI processes, while the optimized algorithm is extremely well balanced. In Figure 3 only the MPI communication times are shown for the two accumulation algorithms. Again for the standard algorithm the fluctuations in the MPI communication times are more pronounced compared to the optimized algorithms. It is not possible to perfectly balance the MPI communication, because the number of boundary vertices in a subdomain cannot be changed, and thus the amount of data that has to be exchanged varies across all MPI processes. Still, for the standard algorithm the imbalances in the CPU execution times amplify the imbalances in the collective MPI communication routines. Finally, Figure 4 shows the total runtimes of the accumulation routines for all MPI processes. The optimized accumulation algorithm is overall about three times faster than the standard accumulation algorithm.

As a last topic we look at the performance of the optimization algorithm in the context of the algebraic multigrid preconditioner with respect to the value of the functional J on different multigrid levels. Table 3 gives an overview of the effectiveness of the optimization algorithm on different multigrid levels and for different numbers of processors. Interestingly, for 6 processors the optimization

TABLE 4. The global number of boundary vertices for different levels of the AMG algorithm and different numbers of processors.

	Processors			
Level	6	12	24	48
1	29461	44581	62756	83738
2	14003	20948	29225	38326
3	4721	6874	9305	11609
4	1285	1793	2180	2413
5	306	347	358	360
6	60	61	65	58
7	15	13	14	12
8	5	5	4	5
9	2	1	2	2
10	1		1	1

TABLE 5. The global sum of all local boundary vertices for different levels of the AMG algorithm and different numbers of processors.

	Processors			
Level	6	12	24	48
1	59482	90475	128241	172715
2	28746	43686	62218	83765
3	10068	15243	21725	29178
4	2990	4631	6470	8827
5	885	1347	1994	2953
6	268	452	793	1196
7	88	145	286	497
8	30	60	90	240
9	12	12	48	96
10	6		24	48

algorithm always attains the global minimum. The effectiveness slightly decreases with an increasing number of processors. Still, compared to the global number of boundary vertices $N = |B|$ given in Table 4 or the global sum of the local boundary vertices $M := \sum_{p=1}^{P} |B_p|$ given in Table 5 the deviation from the global minimum is small. Furthermore, Table 6 gives the average number of MPI processes sharing a boundary vertex.

5. Discussion

In the presented work we show that in the context of parallel linear algebra algorithms, imbalanced subdomain boundary areas can induce not only imbalances in the computational load across MPI processes during the boundary data accumulation, but also can cause additional waiting times for collective MPI communication routines. We discuss a standard and an optimized accumulation algorithm, while the latter introduces balanced sets of master MPI processes for the boundary vetrices, that are used for the boundary data accumulation. Furthermore we present a parallel optimization algorithm for the master MPI process selection and show the effectiveness of the algorithm in a PCG-AMG solver benchmark for an elliptic model problem. The Parallel benchmarks also show that the optimization algorithm

TABLE 6. The average number of processors sharing a boundary vertex for different levels of the AMG algorithm and different numbers of processors.

	Processors			
Level	6	12	24	48
1	2.02	2.03	2.04	2.06
2	2.05	2.09	2.13	2.19
3	2.13	2.22	2.33	2.51
4	2.33	2.58	2.97	3.66
5	2.89	3.88	5.57	8.20
6	4.47	7.41	12.20	20.62
7	5.87	11.15	20.43	41.42
8	6.00	12.00	22.50	48.00
9	6.00	12.00	24.00	48.00
10	6.00		24.00	48.00

does not have a significant impact on the solver setup time, while the runtime for the optimized accumulation routine is significantly reduced, leading to a nearly 50 percent runtime improvement of the PCG-AMG solver benchmark with 48 MPI processes, compared to the standard approach.

The research was supported by the Austrian Science Fund FWF project SFB F032 "Mathematical Optimization and Applications in Biomedical Sciences".

REFERENCES

[1] Baker, A.H., Schulz, M., Yang, U.M.: On the performance of an algebraic multigrid solver on multicore clusters. In: J.M.L.M. Palma, M.J. Daydé, O. Marques, J.C. Lopes (eds.) VECPAR, *Lecture Notes in Computer Science*, vol. 6449, pp. 102–115. Springer (2010)

[2] Briggs, W.L., Henson, V.E., McCormick, S.F.: A Multigrid Tutorial. SIAM Books, Philadelphia (2000). Second edition

[3] Dekker, A., Aarts, E.: Global optimization and simulated annealing. Mathematical Programming **50**, 367–393 (1991)

[4] Geimer, M., Wolf, F., Wylie, B.J., Abraham, E., Becker, D., Mohr, B.: The Scalasca performance toolset architecture. Concurrency Computat.: Pract. Exper. **22**(6), 702–719 (2010)

[5] Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message Passing Interface. The MIT Press, Cambridge (1999)

[6] Haase, G.: A parallel AMG for overlapping and non-overlapping domain decomposition. Electronic Transactions on Numerical Analysis (ETNA) **10**, 41–55 (2000)

[7] Haase, G., Kuhn, M., Reitzinger, S.: Parallel AMG on distributed memory computers. SIAM SISC **24**(2), 410–427 (2002)

[8] Karypis, G., Kumar, V.: MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. `http://www.cs.umn.edu/~metis` (2009)

[9] Knupp, P.M.: Applications of mesh smoothing: copy, morph, and sweep on unstructured quadrilateral meshes. Int. J. for Numerical Methods in Engineering **44**(1), 37–45 (1999)

[10] Liebmann, M.: Efficient PDE solvers on modern hardware with applications in medical and technical sciences. Ph.D. thesis, University of Graz, Austria (2009)

[11] Meurant, G.: The Lanczos and Conjugate Gradient Algorithms, *Software, Environments, and Tools*, vol. 19. SIAM, Philadelphia (2006)

[12] Mitchell, L., Bishop, M., Hoetzl, E., Neic, A., Liebmann, M., Haase, G., Plank, G.: Modeling Cardiac Electrophysiology at the Organ Level in the Peta FLOPS Computing Age, pp. 407–410. AMER INST PHYSICS (2010)

[13] Neic, A., Liebmann, M., Haase, G., Plank, G.: Algebraic multigrid solvers on clusters of CPUs and GPUs. In: K. Jónasson (ed.) PARA (2), *Lecture Notes in Computer Science*, vol. 7134, pp. 389–398. Springer (2012)

[14] Neic, A., Liebmann, M., Hoetzl, E., Mitchell, L., Vigmond, E., Haase, G., Plank, G.: Accelerating cardiac bidomain simulations using graphics processing units. Biomedical Engineering, IEEE Transactions on **59**(8), 2281 –2290 (2012). DOI 10.1109/TBME.2012.2202661

[15] Pechstein, C.: Finite and Boundary Element Tearing and Interconnecting Solvers for Multiscale Problems, *Lecture Notes in Computational Science and Engineering*, vol. 90. Springer (2013)
[16] Plank, G., Liebmann, M., Weber dos Santos, R., Vigmond, E., Haase, G.: Algebraic multigrid preconditioner for the cardiac bidomain model. IEEE Transactions on Biomedical Engineering **54**(4), 585–596 (2007)
[17] van Laarhoven, P., Aarts, E.: Simulated Annealing. Kluwer Academic Press, Dordrecht (1987)
[18] Wohlmuth, B.I.: A mortar finite element method using dual spaces for the lagrange multiplier. SIAM J. Numer. Anal **38**, 989–1012 (1998)
[19] Wolf, F.: Scalasca. In: D. Padua (ed.) Encyclopedia of Parallel Computing, 1 edn., pp. 1775–1785. Springer (2011)