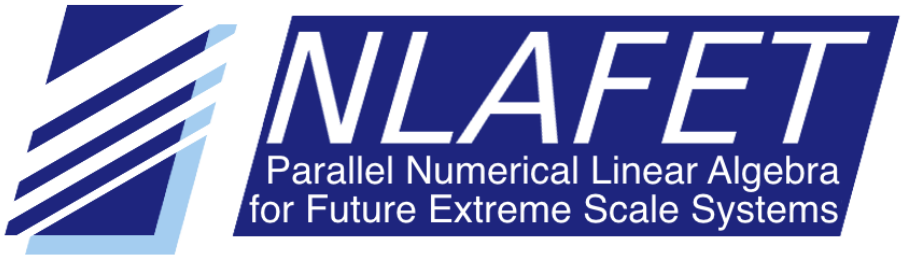


H2020–FETHPC–2014: GA 671633

D3.6

Algorithm design for hybrid methods

October 2017

Document information

Scheduled delivery	2017-10-31
Actual delivery	2017-10-31
Version	1.0
Responsible partner	STFC

Dissemination level

PU — Public

Revision history

Date	Editor	Status	Ver.	Changes
2017-07-10	Iain Duff	Draft	0.1	First draft to determine sections and general framework
2017-10-09	Iain Duff	Draft	0.2	Draft for internal review
2017-10-30	Iain Duff	Version 1	1.0	As sent to Umeå

Author(s)

Iain Duff, STFC
Philippe Gambron, STFC
Florent Lopez, STFC

Internal reviewers

Pierre Blanchard, UNIMAN
Jan Papež, INRIA
Mawussi Zounon, UNIMAN

Contributors

Philippe Leleux, CERFACS, ENSEEIHT-IRIT, France
Daniel Ruiz, ENSEEIHT-IRIT, France

Copyright

Acknowledgements

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under the grant agreement number 671633.

Table of Contents

List of Figures

List of Tables

1 Introduction

The *Description of Action* document states for Deliverable D3.6:

> "*D3.6 Algorithm design for hybrid methods*
>
> *Report on partitioning techniques and performance bottlenecks for hybrid methods including block Cimmino. Analysis of methods for saddle-point and overdetermined systems.*"

This deliverable is in the context of Task 3.4 (Hybrid Direct-Iterative Methods).

To fix our notation, our target is to solve the system

$$Ax = b, \tag{1}$$

where A is a sparse matrix of dimensions $m \times n$. For the matrix A, we only store coefficients that can be nonzero and call these entries. It is possible that some entries might have the numerical value zero either because of numerical cancellation or because we have a set of matrices where an entry is sometimes nonzero but sometimes zero. This might happen, for example, if the matrix is the Jacobian of a nonlinear problem. The entry in row i and column j is denoted by a_{ij}. The right-hand side vector b is of length m and the solution vector x is of length n. In this deliverable, we consider vectors x and b as dense. The methods that we use in this deliverable for solving equation (1) are hybrid methods. Hybrid methods combine direct and iterative methods in order to solve systems larger than can be easily solved with a direct method alone. This means that direct methods need only be used on a subproblem. We discuss this in more detail in Sections 2 and 3.

As we describe in Section 2 there are various hybrid methods and and some can be considered as sophisticated preconditioners of the kind investigated in Task 4.3. Here we will concentrate on block projection methods and, in particular, on the block Cimmino method which we discuss in Section 3. A very important aspect of this method concerns the partitioning of the system to provide parallelism and to accelerate convergence. We consider this in Sections 4 and 5.

The block Cimmino approach is very flexible and we present some preliminary remarks on how it might be used to solve saddle-point problems and on rectangular systems, including least-squares problems, in Sections 6 and 7.

We then describe experiments on partitioning and give some preliminary results on the performance of the code in Section 8.

Finally, we present some conclusions and sketch our future workplan in Section 9.

2 Hybrid methods

The algorithms and code that we are discussing in this deliverable are an example of a *hybrid method* for the solution of sparse equations. The term "hybrid" is used in many contexts, for example hybrid programming combines various elements of parallel support packages (multi-threading, OpenMP, and MPI). The term hybrid Fortran defines a Python-based preprocessor that uses OpenMP on CPUs and CUDA Fortran on GPUs. Of course, hybrid computing refers to computing in a heterogeneous environment with, for example, CPUs and GPUs. In our context, the term hybrid has been used since the early 1980s and refers to methods that combine direct and iterative methods to solve a set of linear equations.

There are several examples of hybrid methods in this context. In multigrid methods, one commonly uses a direct method as a coarse grid solver but the overall method is clearly iterative.

In domain decomposition approaches, a direct method is often used on the local subdomains and an iterative method on the interface problem, possibly preconditioned using direct method techniques. There are many examples of codes using this approach, for example MaPHYS (Inria, Bordeaux), PDSLin (LBNL), ShyLU (Sandia, Albuquerque), and HIPS (Inria, Bordeaux and Minnesota). In the lorasc preconditioner in workpackage 4.3, the preconditioners are based on the Schur complement and direct solves are used in each subdomain.

Another possibility is to perform a direct factorization and solution of a nearby problem for use as a preconditioner, an example of this being the code PSPIKE from Purdue.

Finally, we have the class of block iterative methods where a direct solver is used on the subblocks and this factorization is effectively used to precondition the iterative solver through a block Jacobi or block Gauss–Seidel algorithm. The method that we are using in this deliverable is from this class and can be regarded as an accelerated block Cimmino method. We discuss this in more detail in the next section.

3 Block Cimmino method

We now describe the block Cimmino (BC) method for solving linear systems such as (1). We first consider the case when A is square or underdetermined of dimensions dimensions m by n with $m \leq n$. The overdetermined case is discussed in Section 7. The matrix A can be partitioned as:

$$\begin{pmatrix} A^1 \\ A^2 \\ . \\ . \\ . \\ A^P \end{pmatrix} x = \begin{pmatrix} b^1 \\ b^2 \\ . \\ . \\ . \\ b^P \end{pmatrix} \tag{2}$$

and the algorithm computes a solution iteratively from an initial estimate $x^{(0)}$ according to:

$$u^i = A^{i^+}\left(b^i - A^i x^{(k)}\right) \quad i = 1, \ldots, P \tag{3}$$

$$x^{(k+1)} = x^{(k)} + \omega \sum_{i=1}^{P} u^i, \tag{4}$$

where A^{i^+} is the Moore-Penrose generalized inverse of A^i.

We note that the set of P equations are totally independent and can be solved in parallel although all systems need to be solved before the update of x in the second equation can be completed.

When solving the P subproblems in equation (3), we note that the shape of these subproblems is as shown in Figure 1, where $r^i = b^i - A^i x^{(k)}$. We have a choice of methods for solving these equations. We could use an orthogonal factorization of the rectangular systems but that can be quite costly and require a lot of storage for the factors. Another possibility is to use the normal equations with the coefficient matrix $A^i A^{i^T}$ but that can

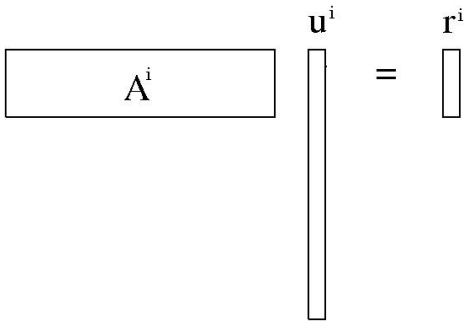


Figure 1: Shape of subproblems.

cause numerical stability problems and the matrix can be quite dense. To get a good balance between stability and preservation of sparsity we choose to solve the equations

$$A^i u^i \;=\; r^i,$$

by direct methods using an augmented system of the form

$$\begin{pmatrix} I & A^{i^T} \\ A^i & 0 \end{pmatrix} \begin{pmatrix} u^i \\ v^i \end{pmatrix} \;=\; \begin{pmatrix} 0 \\ r^i \end{pmatrix}. \tag{5}$$

These systems are symmetric indefinite so we need a direct code that can accommodate numerical pivoting. Since we also want to perform each solution in parallel, we use the parallel direct solver MUMPS in the first instance.

Notice that, in common with most hybrid methods, by setting a block size larger than the original system dimension, we would just be performing a direct solution, albeit not very efficiently. From this viewpoint, we see that the hybrid approach can be considered more as an extension to direct methods rather than just as a preconditioning of an iterative method.

The iteration equations (3-4) can be written as:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \omega \sum_{i=1}^{P} A^{i^+} \left(b^i - A^i x^{(k)} \right) \\ &= \left(I - \omega \sum_{i=1}^{P} A^{i^+} A^i \right) x^{(k)} + \omega \sum_{i=1}^{P} A^{i^+} b^i \\ &= Q x^{(k)} + \omega \sum_{i=1}^{P} A^{i^+} b^i. \end{aligned}$$

We can write the fixed point iteration as

$$Hx \;=\; \xi, \tag{6}$$

where $H = I - Q$. Then the matrix $H = \omega \sum_{i=1}^{P} A^{i^+} A^i$ is a sum of projection matrices and is thus positive definite. Therefore the system (6) can be solved by the conjugate gradient method. We note that, since $\xi \;=\; \omega \sum_{i=1}^{P} A^{i^+} b^i$, the parameter ω appears on both sides of equation (6) so we can arbitrarily set it to one.

The convergence of the block Cimmino method depends on the angles between the subspaces determined by the partitioning in equation (2). Clearly if the subspaces were

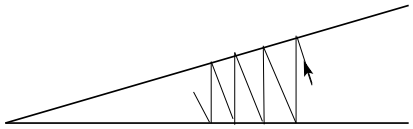


Figure 2: Convergence when angle between subspaces is small.

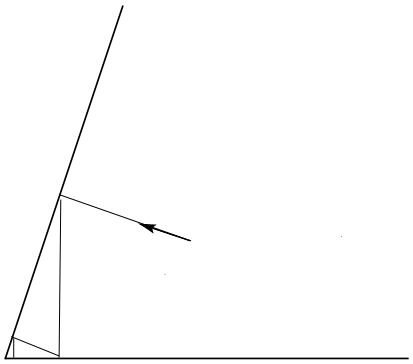


Figure 3: Convergence when angle between subspaces is large.

orthogonal, for example if the columns could be permuted so that the partitioned matrix was block diagonal, then the solution to the overall problem would just be a direct sum of the solution to the subproblems and no iterations would be required. In fact, the cosine of the angles between subspaces is given by the singular values of the matrices $A^i A^{jT}$ that we would like to be close to zero. We thus see the major role that is played by the partitioning algorithm which is why we spend time discussing it in this deliverable.

There are two ways that ill-conditioning can affect the solution when using the block Cimmino algorithm.

- One is in the direct solver. The systems being solved are symmetric indefinite problems where ill-conditioning can cause most sparse direct methods to behave poorly or unpredictably.
- The other way is between the blocks where the block Cimmino algorithm can require many iterations if the subspaces are far from orthogonal.

As a simple illustration of this second effect we illustrate the convergence of a simple Cimmino iteration between two planes where we see that if we widen the angle from that shown in Figure 2 to that shown in Figure 3 then the number of iterations substantially decreases.

In the work of [18], several experiments were performed comparing the block Cimmino code with the direct sparse solver, MUMPS. Usually, MUMPS was faster but required considerably more memory than block Cimmino. However, sometimes block Cimmino was faster and we show, in Table 1, some of Zenadi's results on test matrices from Tim Davis' collection[1] [5]. The reason for the `F` for Cage14 is that MUMPS did not have sufficient memory available to factorize the matrix. Indeed, as we see in Table 2, the block Cimmino method requires far less memory than MUMPS because the direct solver is only used on subproblems. This is true in general since the performance and memory

[1] `https.tamu.edu`

Problem	Order	Entries	Factorization times		
			MUMPS	**BC**	**CG its**
Cage13	445,315	7,479,343	76	2.8	4.1
Cage14	1,505,785	27,130,349	F	5.8	14.5
Hamrle3	1,447,360	5,514,242	208.4	4.7	300.5

Table 1: Times in seconds for MUMPS and block Cimmino (BC) using 64 mpi-processes and 16 threads per mpi-process.

required by a sparse direct factorization is not linear in problem size so the sum of the costs for the factorization of the smaller problems will usually be less than the cost for the original problem.

Problem	Memory per node	
	MUMPS	BC
Cage13	2.8 GB	37 MB
Cage14	30 GB	102 MB
Hamrle3	462 MB	53 MB

Table 2: Storage required for MUMPS and BC.

An approach that was developed by [6] enforces numerical orthogonality between the partitions by adding extra variables and constraints and extracts a condensed small subsystem that can be reused for efficient further solves (similar to Schur complement techniques). This method was defined in [6] as the ABCD solver, standing for **A**ugmented **B**lock **C**immino **D**istributed solver. That is the system $Ax = b$ is augmented to

$$\begin{bmatrix} A & C \\ B & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix} \tag{7}$$

where the added matrices are chosen so that the subspaces generated by the partitions are mutually orthogonal and the solution x is unchanged.

4 Preprocessing for block iterative methods

The main preprocessing steps that we consider are based on scaling and partitioning. We assume that the matrix is irreducible (that is cannot be permuted to a non-trivial block triangular form).

There are good algorithms available to partition irreducible matrices to a bordered block diagonal form as shown in Figure 4. If we use this partitioning to define the blocks of rows for the block Cimmino method so that each block on the diagonal defines these blocks, then the block rows are only non-orthogonal because of the overlaps in the border columns. Thus a method that reduces the number of columns in the border could be expected to be beneficial. The market leader for obtaining this form is generally considered to be PaToH [2], but this code is not parallel and is not Open Source so we examine the use of other codes in Section 8.

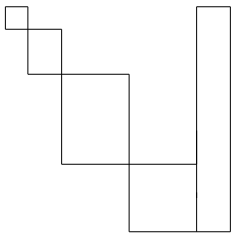


Figure 4: A singly bordered block diagonal form.

Because we are developing robust software for solving general unsymmetric systems, it is crucial to scale the matrix prior to factorization. The market leader here is the HSL code MC64 [8] but again it is not parallel so we use a simpler but still effective scaling given in [1].

5 Hypergraph partitioning

A hypergraph $\mathcal{H}$, is defined by a pair of sets $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ where $\mathcal{V}$ is a set of *vertices* and $\mathcal{N}$ a set of *hyperedges* also called *nets*. Every net $n \in \mathcal{N}$ in this hypergraph is defined as a subset of vertices that is to say $n \subseteq \mathcal{V}$. From this definition it is easy to see that a graph is a special case of a hypergraph where every net contains two elements. We illustrate how to translate the original matrix partitioning problem into a hypergraph partitioning problem using the simple 6-by-6 sparse matrix with the sparsity pattern presented in Figure 5a. From this sparsity pattern we build the hypergraph shown in Figure 5b where vertices correspond to row indices and we define one net per column containing the row indices with nonzero entries in this column. We thus, in our hypergraph, have the vertex set $\mathcal{V} = \{1, 2, 3, 4, 5, 6\}$ and six nets $\mathcal{N} = \{n_1, n_2, n_3, n_4, n_5, n_6\} = \{\{1, 5\}, \{1, 2, 4, 6\}, \{3, 6\}, \{1, 4\}, \{1, 2, 3, 5\}, \{2, 3, 6\}\}$.

Recall, from Section 4, that our aim is to partition A into block-rows such that the number of overlapping nonzero columns between the blocks is minimized. If we look back at our hypergraph model, finding a block-row partitioning corresponds to finding a vertex partition of $\mathcal{H}$ such that each net is involved in as few partitions as possible. This objective is known as a K-way partitioning problem.

The K-way partitioning problem consists in finding a partitioning of $\mathcal{H}$ defined as $\mathcal{P} = \{\mathcal{V}_1, \mathcal{V}_2, .., \mathcal{V}_K\}$ where each part $\mathcal{V}_k$ is a pairwise disjoint nonempty subset of $\mathcal{V}$, while minimizing a given cost function $\mathcal{X}$ also called the *cutsize*. We define $c[n]$ as the cost of any net $n \in \mathcal{N}$ and, for a partitioning $\mathcal{P}$, we define the connectivity of this net λ_n as the number of parts connected by n. A net is said to be *cut* if it connects more than one part. The most commonly used cost functions for computing the cutsize are either

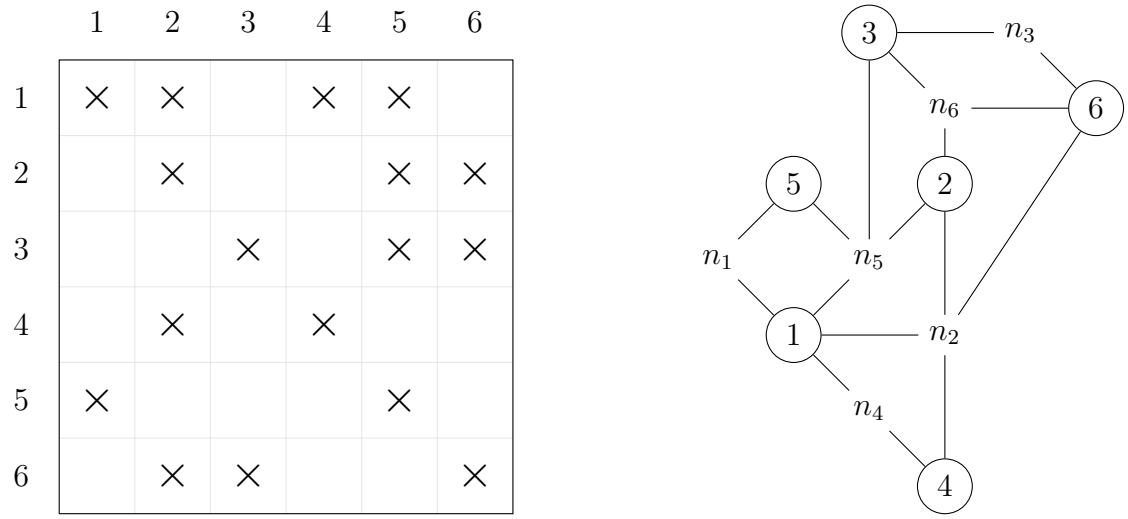
$$\chi_1(P) = \sum_{n \in \mathcal{N}_E} c[n], \tag{8}$$

where $\mathcal{N}_E$ represents the set of nets that are cut, or

$$\chi_2(P) = \sum_{n \in \mathcal{N}_E} (\lambda_n - 1) c[n]. \tag{9}$$

In addition, the cutsize minimization problem can be constrained by a load balancing criterion such as for each part $\mathcal{V}_k$ we impose

$$W_k \leq W_{avg}(1 + \epsilon) \tag{10}$$

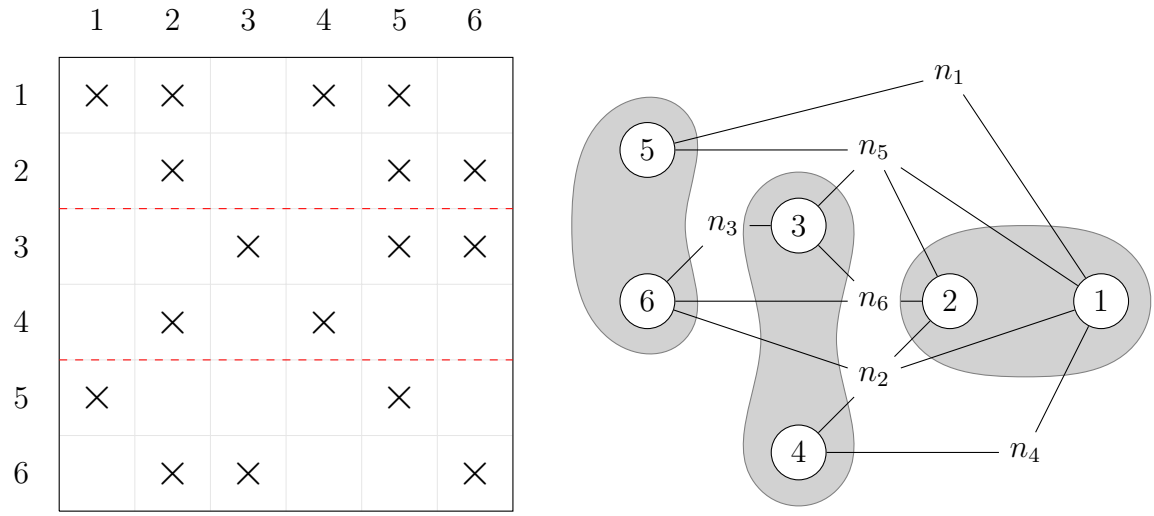


(a) Sparse matrix nonzero pattern. (b) Hypergraph.

Figure 5: Figure 5a illustrates the nonzero pattern of a 6-by-6 sparse matrix and we show in Figure 5b the hypergraph associated with this nonzero pattern. In this hypergraph, each node corresponds to a row and each net (hyperedge) corresponds to a column and connects all the rows that have nonzero entries in this column.

where W_k corresponds to the weight of part $\mathcal{V}_k$, W_{avg} is the average part weight and ϵ is a maximum imbalance ratio allowed. Note that the hypergraph partitioning problem is known to be NP-hard [13].

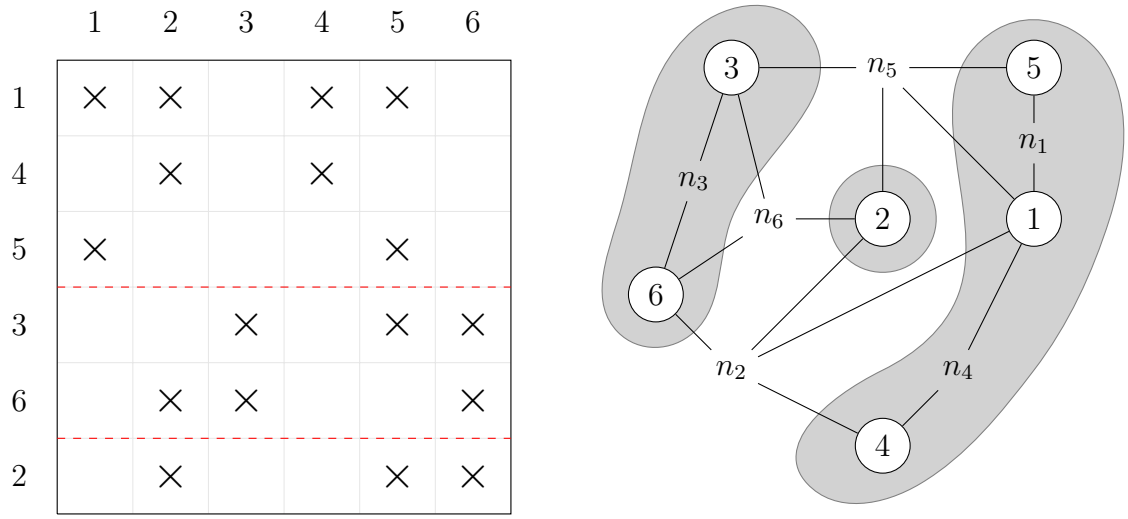
Using our small example, we illustrate in Figure 6a a naive partitioning of the original matrix into three block-rows. The partitioning is $\mathcal{P} = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3\} = \{\{1,2\},\{3,4\},\{5,6\}\}$. Note that, in our experiments, we assign a unit cost to every net and, because we aim to have a similar number of rows in each partition, we assign unit weight to the vertices (that is, $W_k = \text{cardinality}(\mathcal{V}_k)$. In this partitioning, the load balancing is perfect because each partition contains two rows and the cutsize of this partitioning is $\chi_1(P) = 6$ and $\chi_2(P) = 10$. In Figure 7a, we illustrate another



(a) Naive matrix partitioning. (b) Naive Hypergraph partitioning.

Figure 6: Naive matrix partitioning and hypergraph.

choice of partitioning where $\mathcal{P} = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3\} = \{\{1, 4, 5\}, \{3, 6\}, \{2\}\}$. It is easy to see from Figure 7a that the number of net cuts is reduced and thus the amount of overlapping between nonzero columns in the block is limited compared to the previous naive partitioning. We have $\chi_1(P) = 3$ and $\chi_2(P) = 5$. Note that, although we have fewer interactions between the rows across the blocks, this partitioning has a larger load imbalance than the previous one.



(a) Matrix partitioning. (b) Hypergraph partitioning.

Figure 7: Matrix partitioning and hypergraph.

6 Saddle-point problems

Matrices of the form

$$S = \begin{pmatrix} H & A \\ A^T & 0 \end{pmatrix} \tag{11}$$

where H and A are sparse matrices, H is square of order m and A is overdetermined (of dimensions $m \times n$, $m \geq n$) are called saddle-point matrices. These are ubiquitous in scientific computing and arise in constrained problems and mixed finite elements, for example. The matrix S in (11) is sparse, symmetric but indefinite. The structure is really important and most algorithms that ignore this can, on occasion, do spectacularly badly.

If a matrix of this structure is processed by a standard partitioning algorithm like PaToH then the structure will not be fully recognized. Note that, if one partition is all in the last n rows of the matrix S, then it does not matter what the entries are in the last n columns of any of the other blocks, the nearness to orthogonality is obtained from only the first m entries in the rows of the block.

The question that we want to address is whether it is better to constrain the partitioning to avoid including rows from the first m rows and those from the last n rows in the same partition. We could partition the first m rows and the last n rows independently but then there may be more overlap in the first m columns than from a structure-blind approach.

We plan to investigate this during the next period of the NLAFET project.

7 Overdetermined systems

A major positive aspect of the block Cimmino method is that it does not require the matrix in equation (1) to be symmetric nor even square. The method is therefore a candidate for solving both underdetermined and overdetermined systems without the necessity of forming the normal equations. In the keynote paper by Elfving [9], he shows that the algorithm returns the minimum norm solution of a consistent system in the underdetermined case and the least squares solution in the overdetermined case.

In our original formulation and in the code of Zenadi [18] that we use for our experiments in this paper, we use row partitioning of the underdetermined system. This is important since if the matrix is of full rank, that is to say it has full row rank, then the subsystems that are solved by the direct method will have the same property. However, if we perform a row partitioning on an overdetermined matrix even if it is of full rank (column rank) so that the least-squares solution is unique, then the subsystems need not be nonsingular.

We must thus use a column partitioning on the overdetermined matrix that, as shown in [9], goes through in exactly the same way as described for the underdetermined and square cases in Section 3. We have not yet modified the code of [18] to do this, partly because we have not received least-squares problems from our application partners, but we are currently in communication with them to obtain such problems.

8 Preliminary results

In this section we present some results comparing partitioners and some very preliminary results from the block Cimmino code. The matrices that we use are given in Table 3. We use the two smaller matrices to compare the partitioning algorithms, and we run the block Cimmino algorithm on our local parallel platform on the two larger matrices. The matrices are from the SuiteSparse set of test matrices [5].

Matrix	Application	n	nnz
bayer01	chemical process simulation	57,735	275,735
cage10	DNA electrophoresis	11,397	150,645
cage11	DNA electrophoresis	39,082	559,722
cage12	DNA electrophoresis	130,228	2,032,536

Table 3: Matrices used in this study.

Our starting point is the Open Source code of Mohammed Zenadi [18]. We have been porting his code that uses MPI and multi-threading to our machines and have been resolving a few issues, working jointly with colleagues in Toulouse, Philippe Leleux and Daniel Ruiz. The default partitioning method used in the code is PaToH [4]. In the NLAFET project, we are now testing various partitioning approaches since the number of block Cimmino iterations is strongly related to the number of columns in the border of the bordered block diagonal form that we illustrated in Figure 4. For these experiments, we used all the hypergraph codes for which we had easy access and which had not too severe licensing issues. hMETIS [11] is a code from the MeTiS suite and is not Open Source, but we tested it as it is a hypergraph partitioner with good documentation. HSL_MC66 is from HSL and so is also not Open Source but it is local to RAL and so easy to access

and use. It is an instantiation of the MONET code of Blake and Hu [10]. The Zoltan code from Sandia [3] is Open Source and has a parallel version, PHG, so we study this particularly closely as for these reasons it looks the best replacement for PaToH.

We also tried to use the Mondriaan code of Bisseling [14, 17] but, although it does now provide matrix partitionings, its primary goal is the parallel implementation of sparse matrix - vector multiplication, and we found that, although it gave good partitionings, they had a much smaller number of parts than requested by our codes. Indeed, in most cases, only two or four parts were returned. We had trouble downloading the new hypergraph partitioning code, KaHyPar [15], from Karlsruhe due to missing dependencies and are following this up with the authors. However, KaHyPar is only a serial code and, although a parallel version is planned, it is likely to be some time before it is available.

We show, in Figures 8 to 11, some results from four of the codes on the matrices bayer01 and cage10 from Table 3. For the bayer01 matrix, the number of cut edges in Figure 8, that is the number of border columns in the matrix form shown in Figure 4, is quite similar for all codes with Zoltan and PaToH having very similar performance. Another important feature of partitioning codes is that they try to obtain a good balance between the sizes of the parts as this can greatly effect parallel performance, particularly on distributed memory architectures. We see, for the bayer01 matrix in Figure 9, that three of the codes all achieve a balance of less than 1.1 for the ratio of the largest to the average block size. Although the imbalance is significantly more for hMETIS this is partly caused by the interpretation of the input request by the code.

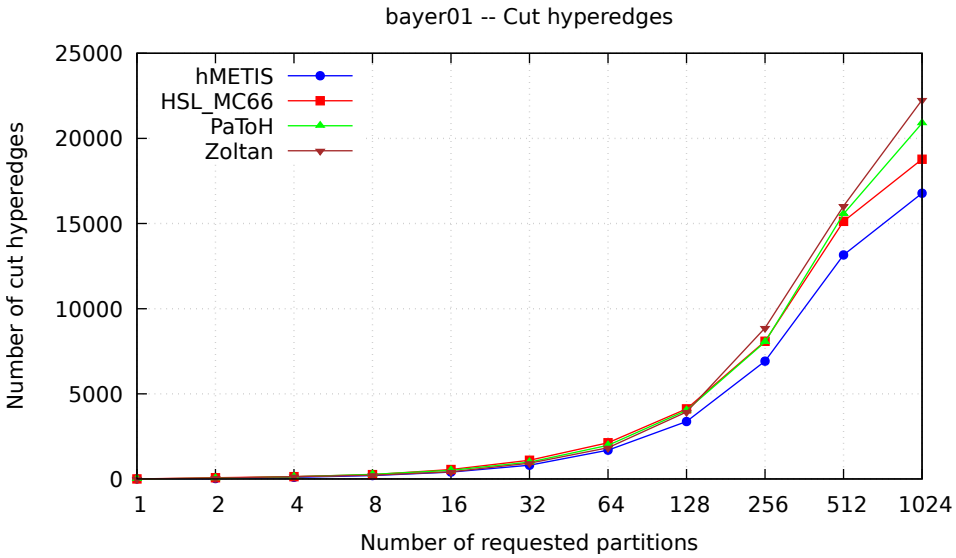


Figure 8: Number of edges in cutset.

A similar relative behaviour of the codes is shown for the cage10 matrix in Figures 10 and 11 although, for all the codes, the number of cut edges increases far more quickly with the number of partitions. This is because there are more denser columns in the cage10 matrix that do need to be included in the cutset to get a good partition.

We have been porting and modifying the code of Zenadi and have obtained good parallelism for some problems. We show the performance of the code on matrix cage12 in Figure 12.

The machine used for these runs is a local heterogeneous machine called SCARF. It

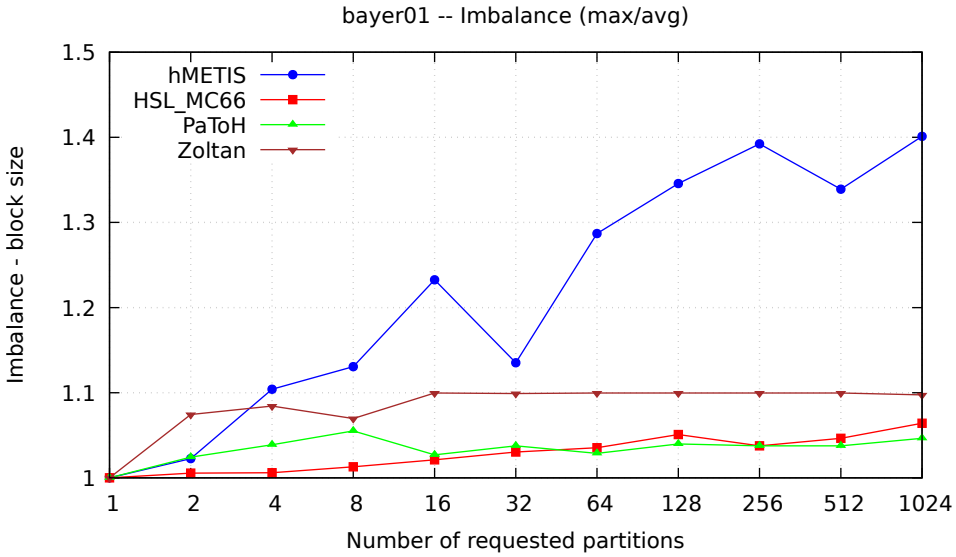


Figure 9: Imbalance of parts in partition.

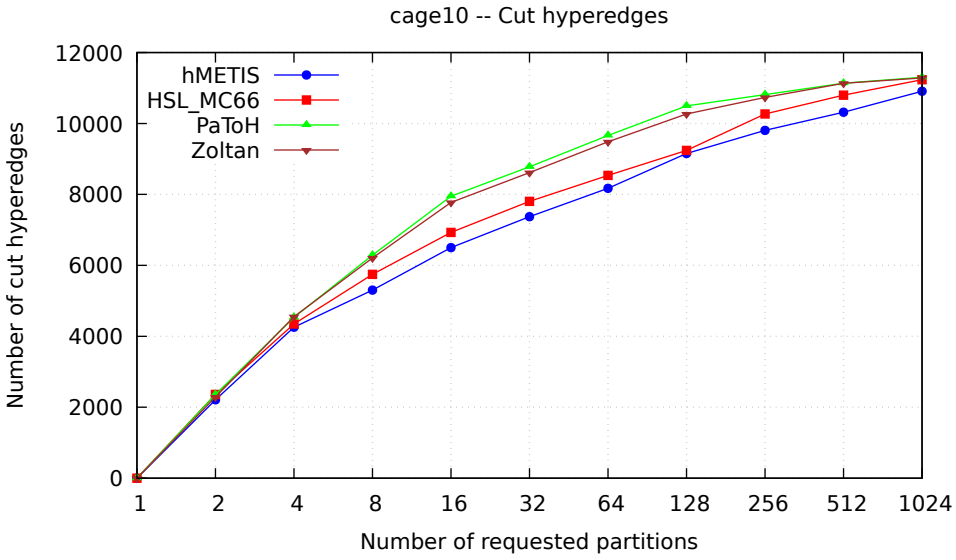


Figure 10: Number of edges in cutset.

has a number of Intel nodes (including some E5-2650, E2660, X5675, X5530, and E5530 nodes)[2]. The actual configuration is determined by the batch scheduler at runtime. In Figure 12 we see the expected good scaling of the iterations (the number of partitions is fixed at 32 for these runs) and reductions in the time spent in the cg solution of equation (6) and the MUMPS factorizations of systems of the form shown in equation (5). We see, in Figure 13, that the number of iterations does not vary significantly when the number of partitions increases although, as expected, the number does increase with the number of partitions.

We note that, in the code of Zenadi and in the earlier paper [6], an algorithm ABCD

[2]See `http://www.scarf.rl.ac.uk/hardware`

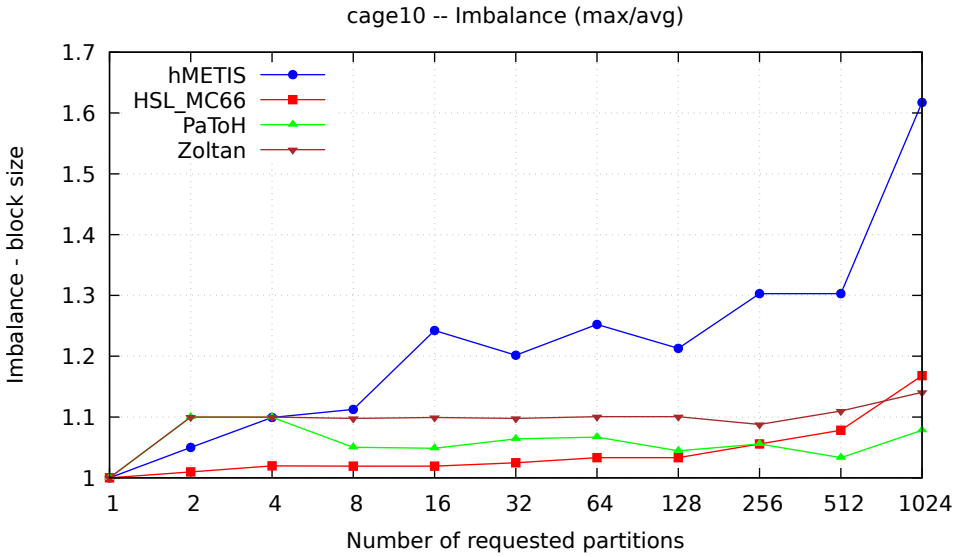


Figure 11: Imbalance of parts in partition.

(Augmented Block Cimmino Distributed) has been developed that totally avoids the interaction between blocks by augmenting them so that they are mutually orthogonal. This yields a pseudo-direct method that should converge in one iteration. We will also be looking at this approach further in the NLAFET project although it arguably is not a hybrid method.

All the partitioners that we considered earlier are largely numerically blind in the sense that they work only with the pattern of the matrix. In a project [7] between ENSEEIHT-IRIT, CERFACS, Strathclyde, and RAL, we are developing algorithms for finding hidden structure in matrices, and we can use this to define a partitioning for the block Cimmino algorithm that identifies the blocks on the diagonal in Figure 4 with clusters obtained from the hidden structure algorithms. The code preprocesses the input matrix to permute it to block triangular form and then first removes highly diagonally dominant rows before invoking the clustering algorithms. Thus we perform experiments on a reduced version of the `cage11` matrix in Table 3 and we show results from this in Figure 14. We see that the partitioning provided by the method in [7], that we denote by "Luce", does give slightly better results than our numerically-blind partitioners. What we have done here is to establish a proof of concept. We will continue to explore this approach with this other team during the next months of the NLAFET project.

Although the standard partitioners are numerically blind, it is possible to put integer weights on edges or nodes, and [16] have used a crude stepwise function to approximate numerical values when using the MeTiS [12] partitioner on the normal equations matrix. They show some improvement over the default approach using PaToH. One of the authors of this paper, Sukru Torun, is joining the EoCoE Centre of Excellence Project in Toulouse. As part of our ongoing collaboration with this CoE, we will be joining forces to develop more powerful partitioners so as to reduce the number of iterations required by the block Cimmino algorithm.

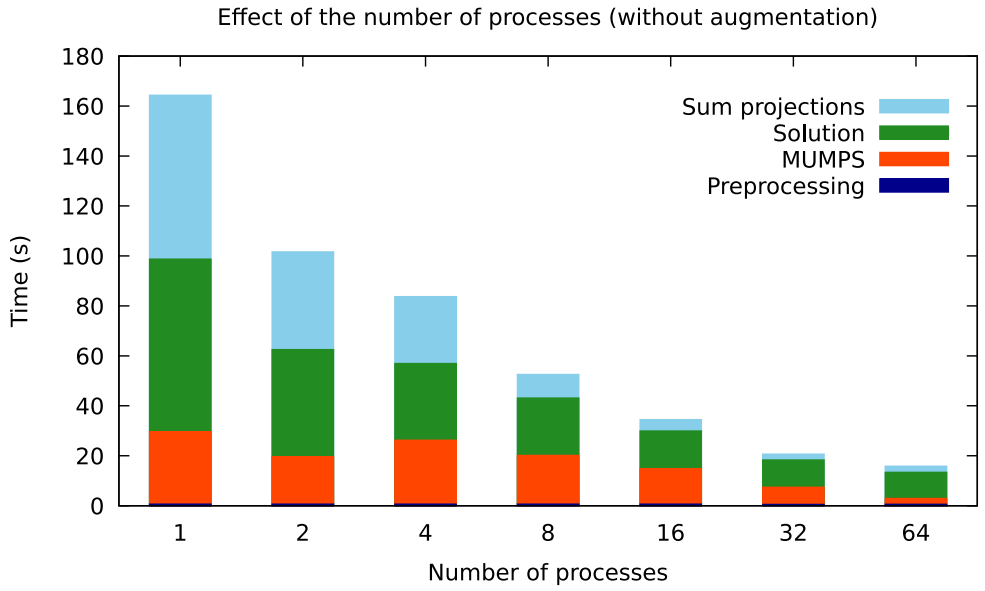


Figure 12: The speedup of block Cimmino for cage12 on the SCARF machine at RAL.

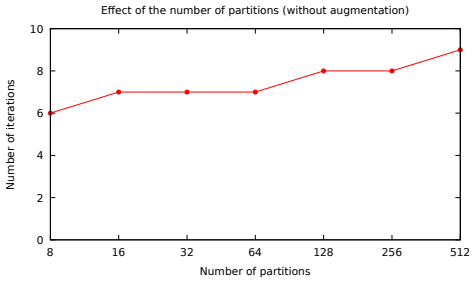


Figure 13: The effect of the number of partitions on the convergence of block Cimmino on a system with matrix cage12.

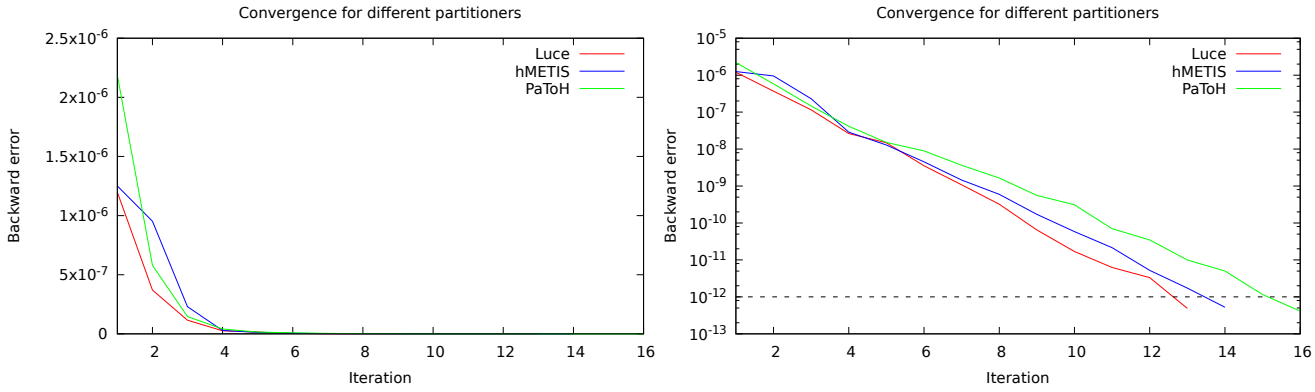


Figure 14: A comparison of partitioners on matrix from cage11.

9 Conclusions and future work

Task 3.4 on hybrid solution techniques is scheduled to continue until the end of the NLAFET project and this current deliverable is mainly to identify the steps that we will perform before the next deliverable for hybrid methods at M36. We now have good familiarity with the code of Zenadi and with appropriate partitioning software, establishing good relationships with the authors of this software. As we discussed in the previous section, we will also be developing, in collaboration with others, numerically-aware partitioners to directly address the convergence of the block Cimmino algorithm.

We will continue our experiments focusing on test problems arising from the applications in our NLAFET project. We also wish to integrate the code with some of the work done in workpackage 3.2, especially since our symmetric indefinite solver is targeting multinode systems. This should marry well with the block approach that we have discussed, as the parallel aspects from the partitioning will map well to distributed memory systems (using MPI) to nicely complement our NLAFET direct solver at the multinode level. In our opinion, this is a viable route to extreme scale exploitation.

Acknowledgements

We would to record that the work on the ABCD code of Mohammed Zenadi was carried out jointly with our colleagues in Toulouse, Daniel Ruiz from ENSEEIHT-IRIT and Philippe Leleux from CERFACS.

References

[1] P. R. Amestoy, I. S. Duff, D. Ruiz, and B. Uçar, *A parallel matrix scaling algorithm*, in High Performance Computing for Computational Science - VECPAR 2008: 8th International Conference, J. M. Palma, P. R. Amestoy, M. Daydé, M. Mattoso, and J. C. Lopes, eds., vol. 5336 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 301–313.

[2] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek, *Permuting sparse rectangular matrices into block-diagonal form*, SIAM Journal on Scientific Computing, 25 (2004), pp. 1860–1879.

[3] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, C. Vaughan, U. Catalyurek, D. Bozdag, 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`.

[4] Ü. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 673–693.

[5] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Softw., 38 (2011), pp. 1–25.

[6] I. S. Duff, R. Guivarch, D. Ruiz, and M. Zenadi, *The augmented block Cimmino distributed method*, SIAM J. Scientific Computing, 37 (2015), pp. A1248–A1269.

[7] I. S. Duff, P. A. Knight, L. Le Gorrec, S. Mouysset, and D. Ruiz, *Uncovering hidden block structure*, Report To appear, ENSEEIHT-IRIT and CERFACS, France and Strathclyde and RAL, UK, 2017.

[8] I. S. Duff and J. Koster, *On algorithms for permuting large entries to the diagonal of a sparse matrix*, SIAM J. Matrix Analysis and Applications, 22 (2001), pp. 973–996.

[9] T. Elfving, *Block-iterative methods for consistent and inconsistent linear equations*, Numerische Mathematik, 35 (1980), pp. 1–12.

[10] Y. F. Hu, K. C. F. Maguire, and R. J. Blake, *A multilevel unsymmetric matrix ordering for parallel process simulation*, Computers in Chemical Engineering, 23 (2000), pp. 1631–1647.

[11] G. Karypis and V. Kumar, hMeTiS – *A Hypergraph Partitioning Package. Version 1.5.3.*, University of Minnesota, Nov. 1998.

[12] G. Karypis and V. Kumar, MeTiS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0*, University of Minnesota, Sept. 1998.

[13] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, Inc., New York, NY, USA, 1990.

[14] D. M. Pelt and R. H. Bisseling, *A medium-grain method for fast 2d bipartitioning of sparse matrices*, in Proceedings IEEE International Parallel & Distributed Processing Symposium 2014, IEEE Press, 2014, pp. 529–539.

[15] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz, k-*way hypergraph partitioning via* n-*level recursive bisection*, in 18th Workshop on Algorithm Engineering and Experiments, (ALENEX 2016), 2016, pp. 53–67.

[16] F. S. Torun, M. Manguoglu, and C. Aykanat, *A novel partitioning method for accelerating the block Cimmino algorithm*, Tech. Rep. arXiv/2044577, Department of Computer Engineering, Kilkent and METU, Ankara, Turkey, 2017.

[17] B. Vastenhouw and R. H. Bisseling, *A two-dimensional data distribution method for parallel sparse matrix-vector multiplication*, SIAM Review, 47 (2005), pp. 67–95.

[18] M. Zenadi, *The solution of large sparse linear systems on parallel computers using a hybrid implementation of the block Cimmino method.*, Thése de Doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, December 2013.