Deliverable D2.1b : MUMPS Version 2.0
A MUltifrontal Massively Parallel Solver

January 23, 1998

Introduction

Deliverable D2.1b consists of a Fortran 90 implementation of a distributed multifrontal code using MPI. It is available through the PARASOL interface. This code is Version 2.0 of MUMPS (MUltifrontal Massively Parallel Solver) and is described in more detail in the Report TR/PA/98/02 that can be considered to be an Appendix to this document.

Since this is the second report, we do not give such a detailed background to multifrontal methods that we did formerly [7], but we nevertheless open with a short discussion, in Section 1, that serves to introduce terms used later in this document.

We then, in Section 2, describe the main features of Version 2.0 of the MUMPS code and discuss the Fortran 90 interface in Section 3. We show how Version 2.0 of MUMPS can be used within the PARASOL interface in Section 4. It is this version that we have distributed to the partners in the Project. Finally, in Section 5, we comment on some enhancements that are planned for future releases of the code.

1 The multifrontal method

MUMPS is a code for distributed memory parallel computers that solves sparse sets of linear equations using a multifrontal method, which is a direct method based on the **LU** factorization of the coefficient matrix. We refer

the reader to our earlier papers [1, 2, 3] for full details of this technique. Version 2.0 of MUMPS solves the system

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

where $\mathbf{A}$ is unsymmetric.

Although we do not wish to describe the multifrontal method in any detail in this document, we will now briefly examine some of the features that are important to our subsequent discussion of the MUMPS code.

The structure of the coefficient matrix is first *analysed* to determine an ordering that, in the absence of any numerical pivoting, will preserve sparsity in the factors. In Version 2.0 of MUMPS, an approximate minimum degree ordering strategy is used on the symmetrized pattern $\mathbf{A} + \mathbf{A}^T$, and this analysis phase produces both an ordering and an assembly tree. The assembly tree is then used to drive the subsequent numerical factorization and solution phases. At each node of the tree, a dense submatrix (called a *frontal matrix*) is assembled using data from the original matrix and from the sons of the node. Pivots can be chosen from within a submatrix of the frontal matrix (called the *pivot block*) and eliminations performed. The resulting factors are stored for use in the solution phase and the Schur complement (the *contribution block*) is passed to the father node for assembly at that node. In the numerical factorization phase, the tree is processed from the leaf nodes to the root (if the matrix is reducible, we have a forest, and each component tree of the forest will be treated similarly and independently). The subsequent forward and backward substitutions during the solution phase process the tree from the leaves to the root and from the root to the leaves, respectively. A crucial aspect of the assembly tree is that it defines only a partial order for the factorization since the only requirement is that a son must complete its elimination operations before the father can be fully processed. It is this freedom that enables us to exploit parallelism in the tree (*tree parallelism*).

In the unsymmetric case, handled by Version 2.0 of MUMPS, threshold pivoting is used to maintain numerical stability so that it is possible that the pivots selected at the analysis phase are unsuitable. In the numerical factorization phase, we are at liberty to choose pivots from anywhere within the pivot block (including off-diagonal pivots) but it still may be impossible to eliminate all variables from this block. The result is that the Schur complement that is passed to the father node may be larger than anticipated by the analysis phase and so our data structures may be different from

these forecast by the analysis. This implies that we need to allow dynamic scheduling during numerical factorization, in contrast to the symmetric positive definite case where only static scheduling is required.

A version of the multifrontal code for shared memory computers was developed by Amestoy and Duff [1] and was included in Release 12 of the Harwell Subroutine Library [6] as code `MA41`. In this version, control and synchronization were enabled through a centralized pool of work, initialized to the leaf nodes. New nodes were added to the pool when all their sons were processed. Because of the reduction in tree parallelism towards the root of tree, it is also necessary to parallelize the computations within a node (*node parallelism*) and this was accommodated in the shared memory code by having two types of task in the work pool identified by a simple flag. Amestoy and Espirat [5] developed a distributed version of the multifrontal code using PVM but included only tree parallelism. This was the basis for Version 1.0 of MUMPS that was released in May 1997.

In the current version of MUMPS (Version 2.0), both tree and node parallelism are exploited, and we distribute the pool among the processors, but our model still requires an identified host node to perform the analysis phase, distribute the incoming matrix, collect the solution, and generally oversee the computation. In the context of PARASOL, we thus support either the host-node model or the hybrid-host model. All routines called by the user for the different steps are SPMD, and the distinction between the host and the other processors is made by the MUMPS code.

2 Description of Version 2.0 of MUMPS

The MUMPS Version 2.0 code is organized with a designated host node and other processors as follows:

1. Analysis. The host computes an approximate minimum degree ordering and performs symbolic factorization. A mapping of the multifrontal tree is then computed, and symbolic information is passed from the host to the other processors. Using this information, the processors estimate the memory necessary for factorization/solve.

2. Factorization. The host sends appropriate entries of the original matrix to the other processors, that are responsible for the numerical factorization. The numerical factorization on each frontal matrix

is conducted by a *master* processor (determined by the analysis phase) and zero or more *slave* processors (determined dynamically) as discussed later in this section. Each processor allocates an array for contribution blocks and factors; the latter should be kept for the solve.

3. Solve. The right-hand side is broadcast from the host to the other processors. These processors compute the solution using the (distributed) factors computed during step 2, and the solution is assembled on the host.

For an efficient and more scalable parallelism on general matrices, the elimination of frontal matrices near the root of the tree has to be parallelized as was, for example, done in the shared memory version of this code. Whereas the previous version of MUMPS (Version 1.0) only used tree parallelism, Version 2.0 of MUMPS also exploits node parallelism; this is done by the introduction of Type 2 and Type 3 nodes, as defined below.

We consider the assembly tree of Figure 1 where, instead of single nodes as the leaves, there are subtrees whose constituent nodes have frontal matrices of small order. Each subtree is processed by a single processor, to avoid communication at that stage. This mapping of subtrees to processors is performed by the analysis phase. For large problems, there will be more subtrees than processors which will aid in the overall load balancing of the computation.

Above the subtrees, there can still be some nodes processed by only one processor. These nodes (as well as nodes inside the subtrees) are called nodes of Type 1.

Consider a typical frontal matrix in the tree in which there are NPIV pivots to eliminate (that is, the pivot block has order NPIV) and NCB rows to update (that is, the order of the frontal matrix is NPIV+NCB). A node is determined to be of Type 2 if NPIV and NCB are large enough. The partial factorization process is then parallelized with the first NPIV rows on one processor, called the *master of the node* and the NCB rows distributed among other processors (called the *slaves of the node*). For instance, in the Type 2 node on the right of Figure 1, P3 is the master, and P0, P1, and P2 are the slaves.

A pipelined factorization is used, and updates on the contribution blocks are performed in parallel. In our implementation, the assembly process is also fully parallel.

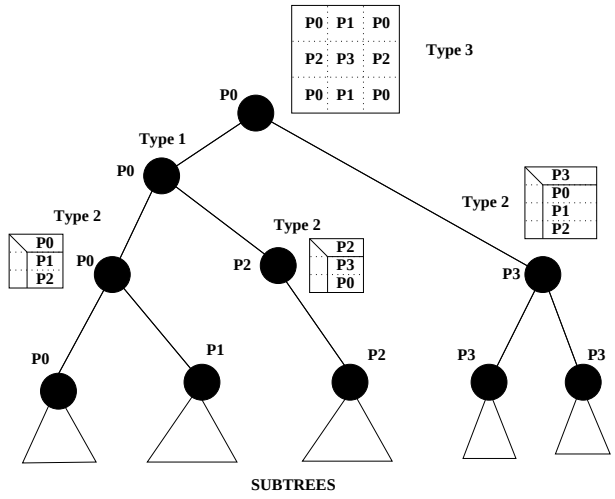

Figure 1: Distribution of the computations of a multifrontal tree

At the root node, a full LU factorization is performed. If the size of the root node is deemed large enough, the root node is said to be of Type 3, and is factorized using ScaLAPACK. An earlier version assembled the root node as a full matrix and redistributed it. In the current version, the assembly of the root node is directly distributed in a 2D cyclic grid and is now completely parallel.

Other characteristics of the code include :

- Distribution of the original matrix to the processors according to a mapping determined by the analysis phase. This allows for much better scalability,
- Scaling of the original matrix, with several possible scaling algorithms,
- A mechanism to handle asynchronous buffered messages that is better than the MPI_BSEND call,
- Numerical pivoting, which means that the code must allow the order of frontal matrices in the tree to increase dynamically,
- Backward error estimate and iterative refinement.

3 Fortran 90 interface

In the Fortran 90 interface to MUMPS, there is a single user callable subroutine called `PSL_MUMPS` that has a single parameter `mumps_par` that holds all data pertinent to a given problem. It is of Fortran 90 derived datatype `STRUC_MUMPS`, viz.

```
TYPE (STRUC_MUMPS) :: mumps_par
CALL PSL_MUMPS( mumps_par )
```

This derived datatype, `STRUC_MUMPS`, has many components, only some of which are of interest to the user. The other components are internal to the package. Some of them must only be defined on the host. Others must be defined on all processors.

The interface to MUMPS consists in calling the subroutine `PSL_MUMPS` with the appropriate parameter settings in the components of its argument. A `mumps.h` file containing a definition of the structure is available and should be included in the program to define the derived data type. An example of how to use the Fortran 90 interface is given Figure 2.

The main control on the package is effected through the parameter mumps_par%JOB that should be set to –1 to initialize structures and set processor ranks with mumps_par%MYID, and to –2 to deallocate them at the end of the computation. Other values of mumps_par%JOB determine whether all phases (analysis, factorization, and solve) will be performed or only a subset of them.

The input matrix and right-hand side are defined by other components of mumps_par in the following way:

- N is the order of the matrix **A**.
- NZ is the number of entries being input.
- IRN, JCN are integer arrays of length NZ containing the row and column indices, respectively, for the matrix entries.
- ASPK is a double precision array of length NZ. The user must set ASPK(K) to the value of the entry in row IRN(K) and column JCN(K) of the matrix.

- RHS is a double precision array of length N. On entry, RHS(I) must hold the I th component of the right-hand side of the equations being solved. On exit, RHS(I) will hold the I th component of the solution vector.

An example of the use of MUMPS is given Figure 2. Two files have to be included: mpif.h for MPI and mumps.h for MUMPS. The initialization and termination of MPI are performed in the user program via the calls to `MPI_INIT` and `MPI_FINALIZE`.

The package MUMPS must be initialized by calling `PSL_MUMPS` with JOB=–1, then the problem is defined on the host, and the solution is computed with a call to `PSL_MUMPS` with JOB=6, that causes all three phases to be implemented. Finally, a call to `PSL_MUMPS` with JOB=–2 is performed to deallocate data structures used by the instance of the package.

```
      PROGRAM MUMPS
      INCLUDE 'mpif.h'
      INCLUDE 'mumps.h'
      TYPE (STRUC_MUMPS) mumps_par
      INTEGER IERR
      CALL MPI_INIT(IERR)
C Define a communicator for the package.
      mumps_par%COMM = MPI_COMM_WORLD
C  Initialize an instance of the package.
      mumps_par%JOB = -1
      CALL PSL_MUMPS(mumps_par)
C  Define problem on the host (processor 0)
      IF ( mumps_par%MYID .eq. 0 ) THEN
        READ(5,*) mumps_par%N
        READ(5,*) mumps_par%NZ
        ALLOCATE( mumps_par%IRN ( mumps_par%NZ ) )
        ALLOCATE( mumps_par%JCN ( mumps_par%NZ ) )
        ALLOCATE( mumps_par%ASPK( mumps_par%NZ ) )
        ALLOCATE( mumps_par%RHS ( mumps_par%N  ) )
        READ(5,*) ( mumps_par%IRN(I) ,I=1, mumps_par%NZ )
        READ(5,*) ( mumps_par%JCN(I) ,I=1, mumps_par%NZ )
        READ(5,*) ( mumps_par%ASPK(I),I=1, mumps_par%NZ )
        READ(5,*) ( mumps_par%RHS(I) ,I=1, mumps_par%N  )
      END IF
C  Call package for solution
      mumps_par%JOB = 6
      CALL PSL_MUMPS(mumps_par)
C  Solution has been assembled on the host
      IF ( mumps_par%MYID .eq. 0 ) THEN
        WRITE( 6, * ) ' Solution is ',(mumps_par%RHS(I),I=1,mumps_par%N)
      END IF
C  Destroy the instance (deallocate data structures)
      mumps_par%JOB = -2
      CALL PSL_MUMPS(mumps_par)
      CALL MPI_FINALIZE(IERR)
      STOP
      END
```

Figure 2: Example program using Fortran 90 interface to MUMPS

4 PARASOL interface

Integration of Version 2.0 of the MUMPS solver into the PARASOL package uses the PARASOL interface routines as described in [4]. In particular, we supply all the required data exchange routines for communication between the program and a package instance:

- `psl_mumps_contract` - establishing/closing a data exchange session,
- `psl_mumps_what2send` - send an inquiry,
- `psl_mumps_send?data` - send data,
- `psl_mumps_need2recv` - request data, and
- `psl_mumps_recv?data` - receive data.

These routines are used only for data exchange between a user's application (program) and the MUMPS solver (package). The current version of the MUMPS integration into the PARASOL package supports only the `HOST-NODE` execution model (`PSL_MODE = PSL_HOSTNODE`), which is schematically described in Figure 3.

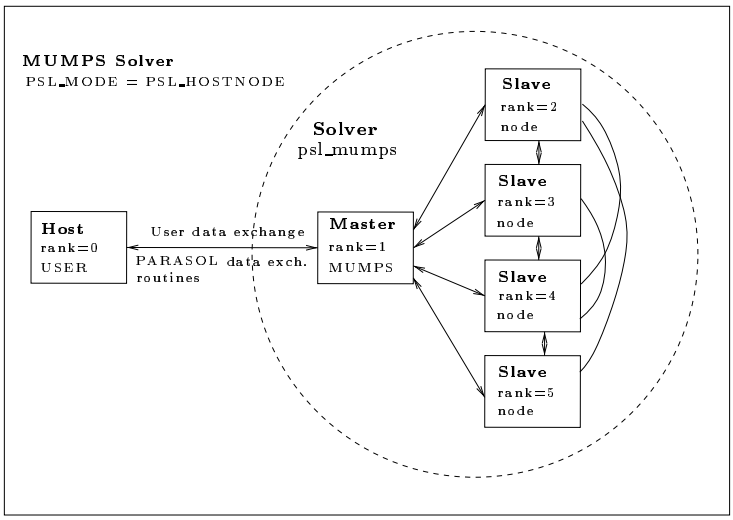

Figure 3: HOST-NODE execution model for MUMPS

The basic data communication requires: sending a matrix (PSL_NAME = PSL_MATRIX), and a right-hand side vector (PSL_NAME = PSL_RHSIDE, and receiving a solution vector (PSL_NAME = PSL_SOLUTION).

The MUMPS solver itself is controlled by the PARASOL control routines:

- psl_mumps_init - instance initialization,
- psl_mumps_end - instance termination,
- psl_mumps_map - mapping routine,
- psl_mumps_endmap- mapping termination,
- psl_mumps_solve - solution routine, and
- psl_mumps_endsolve - solution termination.

Unlike the exchange routines, the control routines make heavy use of the internal structure of the MUMPS code. Their implementation is based on appropriate calls to the subroutine PSL_MUMPS. We use the name "MUMPS code" to refer to the whole set of routines for which PSL_MUMPS serves as a driver. The MUMPS instance data is held in a structure called psl_mumps_i of Fortran 90 derived type STRUC_MUMPS (see psl_mumps_inst.h).

- psl_mumps_init - this routine initializes the MUMPS instance:
  1. creating description of nodes, the current version supports only PSL_MODE = PSL_HOSTNODE and the user has reserved one process (rank = 0),
  2. establishing the instance addressing, and
  3. calling the initialization phase of the MUMPS code, MUMPS function: JOB = -1.
- psl_mumps_end - this routine terminates the instance:
  1. deallocating the resources by calling the termination phase of the MUMPS, MUMPS function: JOB = -2.
- psl_mumps_map - the mapping routine performs the analysis and factorization phases of the MUMPS code:

1. setting some output/diagnostics parameters of the MUMPS code according to the PARASOL control parameters (call `psl_mumps_setout`),
2. receiving on the master the sparse structure of the matrix `n,nz,col,row` and creating a representation required by `psl_mumps` (in `psl_mumps_i` records `n,nz,irn,jcn`),
3. performing the analysis phase of the MUMPS code, MUMPS function: `JOB = 1`,
4. receiving on the master numerical values of matrix entries, and
5. numerical factorization, MUMPS function: `JOB = 2`

- `psl_mumps_solve` - the solution routine calls the solution phase of the MUMPS code:
  1. receiving the right hand side(s), and
  2. solving, MUMPS function `JOB = 3`,
- `psl_mumps_endsolve` - the termination routine sends the solution to the user. Deallocation of the used memory is done in the routine `psl_mumps_end` by calling the MUMPS code with the function `JOB = -2`.

Examples of how to use the PARASOL interface routines for building user-defined applications are described in Section 3.2 of [4]. We refer to this guide for more details about the use of the PARASOL interface. However, the user should keep in mind that the current integration of MUMPS into the PARASOL package supports only *the HOST-NODE execution model* and solution of *unsymmetric sparse (assembled) matrices*. Therefore using other PARASOL attributes will generate error messages.

The integration of the MUMPS code allows a user to run the MUMPS solver from the PARASOL test driver, see [8]. The solver has been incorporated in the directory structure of `ptd.2.0` as follows:

- `./src/ral` - generic PARASOL interface routines for CERFACS/RAL solvers,

- `./src/ral/mumps` - specific PARASOL interface routines for the MUMPS code and the MUMPS code itself.

To build the MUMPS solver the user has to modify `./src/makeconf`, and define `SOL = ral`, `SOL2 = mumps`. The MUMPS code uses routines from ScaLAPACK (and consequently from BLACS) therefore these packages have to be properly installed on the machine and the file `makeconf` should be changed according to the installation details. For more details about how to use the PARASOL test driver we refer to [8].

5 Next steps in the development of MUMPS

Version 2.0 of MUMPS is the first to offer full parallelism at both tree and node level and is the basis for the future developments that we now describe.

One of our first tasks, after the formal release of Version 2.0 at the end of January, will be to test and tune the code against some of the test problems supplied by the PARASOL partners. We plan to present some of these results at the Review meeting in Bonn in February. We anticipate that this will only cause minor internal changes to the code resulting in Version 2.1 that will be released in early spring.

We have presented a roadmap for future development of MUMPS but the precise timing is still under discussion with partners. However, future versions will include:

- the solution of transposed systems and an estimate of the forward error,
- an interface to graph partitioning packages,
- a more efficient code for symmetric positive definite matrices,
- an interface to handle unassembled finite-element problems directly, and
- facilities for holding factors out-of-core.

References

[1] Patrick R. Amestoy and Iain S. Duff. Vectorization of a multiprocessor multifrontal code. *Int. J. of Supercomputer Applics.*, 3:41–59, 1989.

[2] Iain S. Duff and John K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw.*, 9:302–325, 1983.

[3] Iain S. Duff and John K. Reid. The multifrontal solution of unsymmetric sets of linear systems. *SIAM J. Scientific and Statistical Computing*, 5:633–641, 1984.

[4] A. Supalov (editor). PARASOL Interface Specification. Version 2.1. January 9, 1998.

[5] V. Espirat. Développement d'une approche multifrontale pour machines à mémoire distribuée et réseau hétérogène de stations de travail. Rapport de stage de 3ieme année, ENSEEIHT-IRIT, Toulouse, France, 1996.

[6] HSL. *Harwell Subroutine Library. A Catalogue of Subroutines (Release 12).* AEA Technology, Harwell Laboratory, Oxfordshire, England, 1996. For information concerning HSL contact: Dr Scott Roberts, AEA Technology, 552 Harwell, Didcot, Oxon OX11 0RA, England (tel: +44-1235-434988, fax: +44-1235-434136, email: Scott.Roberts@aeat.co.uk).

[7] PARASOL. Deliverable D2.1a: distributed multifrontal code. January 25, 1997.

[8] A. Supalov. The Rutherford-Boeing File Formats and the PARASOL Test Driver. Version 2.1. January 9, 1998.