MUMPS

Intro

- MUMPS = "**MU**ltifrontal **M**assively **P**arallel **S**olver"

- A package for solving systems of linear equations:

$$A x = b$$

  where A is a square sparse matrix

- MUMPS implements a **direct method** based on a multifrontal approach

- If A is not symmetric:

$$A = LU$$

- If A is symmetric:

$$A = LDL^T$$

Main Features of MUMPS

- Solution of the transposed system
- Input of the matrix in assembled format (distributed or centralized) or elemental format
- Iterative refinement
- Scaling of the original matrix
- out-of-core capability
- Computation of a Schur complement matrix

Ordering algorithms

- MUMPS offers several built-in ordering algorithms
- It has a tight interface to some external ordering packages:
    - PORD
    - SCOTCH
    - METIS
- Offers the possibility to input a given ordering

Implementation

- Written in Fortran 90
- Comes with a C interface (we'll construct a simpl C++ version)
- Scilab and MATLAB/Octave serial interfaces
- Uses BLAS, BLACS, and ScaLAPACK libraries
- Shared-/Distributed- Memory Parallelization via MPI
- Distributes the work tasks among the processors
- The ***"host"*** performs most of the analysis, distributes the matrix to the ***"slaves"***, collects the solution

Implementation

- Operation is divided in 3 phases:
  - Analysis
  - Factorization
  - Solution
- Each phase can be performed separately
- Several instances of MUMPS can be handled simultaneously
- The host processor usually performs the analysis, it may or may not participate to the factorization and solve phases

Analysis

- Preprocessing of the system including:
  - Ordering based on the *"symmetrized"* pattern of

    $$A + A^T$$

  - Symbolic factorization
- Parallel or sequential implementation
- Mapping of the multifrontal computational graph
- Estimate the number of operations and memory required
- Let A_{pre}, x_{pre}, b_{pre} denote the preprocessed matrix, unknown, RHS

Factorization

- Direct factorization:

$$A_{pre} = L\ U \text{ or } A_{pre} = L\ D\ L^T$$

- The matrix is first distributed according to the ***elimination tree***

- The actual factorization is a sequence of dense factorizations on ***frontal matrices***

- The elimination tree expresses independence allowing parallelization

- This approach is called ***multifrontal approach***

- After the factorization, the matrice factors are kept distributed

Solution

- Forward elimination step

  $L\,y = b_{pre}$ or $L\,D\,y = b_{pre}$

- backward elimination step

  $U\,x_{pre} = y$ or $L^T x_{pre} = y$

- The RHS b is first preprocessed by the host and broadcast to the working processors

- Forward elimination and Backward substitution (Equation 4) are performed using the distributed factors

- The solution x_{pre} is postprocessed to obtain the solution x

- x is either assembled on the host or kept distributed on the working processors

Example (using C interface)

```
/* Example program using the C interface to the
 *  double precision version of MUMPS, dmumps_c.
 *  We solve the system A x = RHS with A = diag(1 2) and RHS = [1 4]^T
 *  Solution is [1 2]^T */

#include <stdio.h>
#include "mpi.h"
#include "dmumps_c.h"
#define JOB_INIT -1
#define JOB_END -2
#define USE_COMM_WORLD -987654


int main(int argc, char ** argv) {
     DMUMPS_STRUC_C id;
     int n = 2;
     int nz = 2;
     int irn[] = {1,2};
     int jcn[] = {1,2};
     double a[2];
     double rhs[2];
     int myid, ierr;

     ierr = MPI_Init(&argc, &argv);
     ierr = MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

Example (using C interface)

```
/* Define A and rhs */

  rhs[0]=1.0;
  rhs[1]=4.0;
  a[0]=1.0;
  a[1]=2.0;


/* Initialize a MUMPS instance. Use MPI_COMM_WORLD. */

  id.job=JOB_INIT;
  id.par=1;
  id.sym=0;
  id.comm_fortran=USE_COMM_WORLD;
  dmumps_c(&id);


/* Define the problem on the host */


   if (myid == 0)
   {
      id.n = n;
      id.nz = nz;
      id.irn = irn;
      id.jcn = jcn;
      id.a = a;
      id.rhs = rhs;
    }
```

Example (using C interface)

```
#define ICNTL(I) icntl[(I)-1] /* macro s.t. indices match documentation */

   /* No outputs */
   id.ICNTL(1)=-1;
   id.ICNTL(2)=-1;
   id.ICNTL(3)=-1;
   id.ICNTL(4)=0;

   /* Call the MUMPS package. */
   id.job=6;
   dmumps_c(&id);

   id.job=JOB_END;
   dmumps_c(&id); /* Terminate instance */

   if (myid == 0) {
      printf("Solution is : (%8.2f %8.2f)\n", rhs[0],rhs[1]);
   }

   return 0;

}
```

Example (using C++ interface)