pARMS: A Package for the Parallel Iterative Solution of General Large Sparse Linear System * User's Guide

Zhongze Li[†], Daniel Osei-Kuffuor[†], Yousef Saad[†] Masha Sosonkina[‡]

January 13, 2011

1 Introduction

For many large-scale applications, solving large sparse linear systems is the most time-consuming part. The important criteria for a suitable solver include efficiency, robustness, and good parallel performance. The Parallel Algebraic Recursive Multilevel Solver (pARMS) [8] is a suite of distributed-memory iterative accelerators and preconditioners targeting the solution of general sparse linear systems. It adopts a general framework of distributed sparse matrices and relies on the solution of the resulting distributed Schur complement systems.

This version of pARMS was reimplemented. The new version has the follow features:

- It uses an object-oriented design. It is easier to maintain the code and plug in new functions without breaking the existing interfaces.
- It separates parallel preconditioners from local ILU preconditioners. A new implemented parallel preconditioner can be used in conjunction with all existing local ILU preconditioner, and vice versa. Item Entries passed to parms_MatSetValues and parms_VecSetValues are expressed in global numbering, which alleviates the users' programming burden.
- It includes new robust preconditioners, e.g., the ddPQ version of ARMS (ARMS with non-symmetric permutatoins) [10] in ITSOL [1].

2 Installation

The installation involves the following steps:

1. Type 'gunzip -c parms.tar.gz — tar xvf -' or 'tar -xvfz parms.tar.gz' on SGI Altix or linux clusters to uncompress and unbundle the file.
2. Copy conf/makefile.XX into parms/makefile.in. XX stands for the architecture of machines. For example, if you wish to install pARMS on a SGI Altix,
   cp conf/makefile.altix makefile.in
   There are three linux related configuration files available. If you use INTEL compilers, you

*This work was supported in part by NSF under grant NSF/ACI-0305120, and in part by the Minnesota Supercomputing Institute

[†]Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455, {dosei,saad}@cs.umn.edu.

[‡]Ames Laboratory, Iowa State University, Ames, IA 50011, masha@scl.ameslab.gov.

may use makefile.linux.icc. If you use gcc 3.xx or above, you may use makefile.linux.gcc3, otherwise you may use makefile.linux.gcc. If the architecture is not in the list, you may copy makefile.linux.gcc into makefile.in and edit makefile.in.

3. Edit makefile.in.

   (a) Define CC, COPTFLAGS, CFLAGS, CFFLAGS, FC, FFLAGS, LINKER, LINKFLAGS, refer to C/FORTRAN compiler and compiler options, linker and linker options. CFFLAGS refers to compiler option for C/FORTRAN mixed programming. The possible options for CFFLAGS are -DFORTRAN_CAPS, -DFORTRAN_DOUBLE_UNDERSCORE, -DFORTRAN_UNDERSCORE. Those options are used to define the calling convention from C to FORTRAN. If you are on a IBM POWER machine, you do not need add any options to the definition of CFFLAGS. If you are on a linux cluster, a SGI Altix, a SGI Origin, or a HP Alpha, you need add -DFORTRAN_UNDERSCORE. If you use a Cray T3D or Cray T3E machine, you need add -DFORTRAN_CAPS. It is easy to check which option should be used. You need to write a short FORTRAN program check.f as follows:

       ```
       subroutine sub()
       end
       ```

       You may use the FORTRAN compiler on your machine to compile it to an object file check.o. Type

       ```
       nm check.o | awk '{print $3}'
       ```

       If sub_ is displayed, then you use -DFORTRAN_UNDERSCORE. If sub__ is displayed, then you add -DFORTRAN_DOUBLE_UNDERSCORE. If SUB is shown, then you must use -DFORTRAN_CAPS.

       If you use a 32-bit machine, you must add -DVOID_POINTER_SIZE_4 to the definition of CFFLAGS. If you are on a 64-bit machine, you must add -DVOID_POINTER_SIZE_8 instead.

   (b) The version if pARMS to be compiled depends on the -DDBL and -DDBL_CMPLX flags. To compile the complex version of pARMS, use the compile flag -DDBL_CMPLX to compile pARMS by including it in the CFLAGS and FFLAGS variables in makefile.in (in place of the -DDBL flag). To compile the real code, use the -DDBL flag instead.

   (c) Specify the locations of libraries: MPI, LAPACK, BLAS, and math libraries.

4. Type 'make' to build the library libparms.a. You may also type 'make tests' to compile the test programs in the 'examples' directory.

3 Quick Start

The framework of distributed linear systems provides [9] an algebraic representation of the irregularly structured sparse linear systems arising in the Domain Decomposition methods. A typical distributed system arises, e.g., from a finite element discretization of a partial differential equation on a certain domain. To solve such systems on a distributed memory computer, it is common to partition the finite element mesh and assign a cluster of elements representing a physical subdomain to one processor. The typical output of a serial graph partitioning routine is *part*, an array of size n (the number of total vertices). $part[i]$ stores the processor label to which the vertex i belong. Each processor then assembles only the local equations restricted to its assigned cluster of elements. In the case where the linear system is given algebraically, a graph containing vertices that correspond to the rows of the linear system can be partitioned. For both cases, the general assumption is that each processor holds a set of equations (rows of the global linear system) and the associated unknown variables. pARMS solves the distributed system in parallel using flexible GMRES combined with domain decomposition based parallel preconditioners. The code fragment for solving a large sparse linear system is shown in Figure 1.

```
/* Partition the mesh or graph  */
call_partioner(&n, ia, ja, ..., &part);

/* Create a parms_Map object based on the output from a mesh partitioning
   software. The parms_Map object represents how data are distributed across
   processors. The following statement creates a parms_Map object
   based on the output of a serial graph partitioning function. */
parms_MapCreateFromGlobal(&map, n, part, ...).

/* Create a distributed matrix based on the parms_Map created above. */
parms_MatCreate(&A, map);

/* Create solution and right-hand-side vectors based on the parms_Map created.
This version of pARMS simply uses C array pointers - double *rhs, sol;*/
nloc = parms_MapGetLocalSize(map);

PARMS_NEWARRAY(rhs, nloc);
PARMS_NEWARRAY(sol, nloc);

/* Insert entries into the matrix */
parms_MatSetValues(A, ...);

/* After calling the following statement, no entries can be inserted
   into the matrix */
parms_MatSetup(A);

/* Insert entries into the right-hand-side vector. No setup is needed for vectors */
for (i = 0; i < n; i++) {
  parms_VecSetValues(rhs,  1, &i, ...);
}

/* Create a preconditioner object based on the matrix A */
parms_PCCreate(&pc,A);

/* Setup the preconditioner type (eg. RAS) */
parms_PCSetType(pc, PCRAS);

/* Setup the type of local ILU preconditioner (eg. ARMS) */
parms_PCILUType(pc, PCARMS);

/* Setup the preconditioning matrix */
parms_PCSetup(pc);

/* Create a solver object based on the matrix A and the preconditioner pc */
parms_SolverCreate(&solver, A, pc);
```

```
/* Solve the linear system of equations */
parms_SolverApply(solver, rhs, sol);
```

Figure 1: A sample application code fragment

3.1 parms_Map object

The parms_Map object represents how data are distributed across processors. It is the most important object in pARMS. All other types of objects are created based on it. Users must make sure the routines used for creating a parms_Map object are consistent with the graph partitioning routines used for distributing the data. If a serial graph partitioning routine is called, the routine parms_MapCreateFromGlobal must be called to create a parms_Map object. Otherwise, the function parms_MapCreateFromDist must be called. pARMS assumes ParMETIS [6] is used for parallel graph partitioning. A local parms_Map object can be created with parms_MapCreateFromLocal, which indicates which part of the data are fully owned by the local processor. For a user-defined partitioning, a function parms_MapCreateFromPtr is provided to read the partitions from an array pointer provided by the user or partitioner. See the reference guide for more details.

Each vertex i is associated with multiple variables u_i, v_i, p_i for multi-component problems. Partitioning routines split vertices into disjoint sets, and variables associated with each vertex are split accordingly. If vertex i is assigned to processor j, variables associated with vertex i, u_i, v_i, p_i, are also assigned to processor j. There are two approaches in pARMS to number variables:

INTERLACED Variables are numbered in the order of $u_1, v_1, p_1, u_2, v_2, p_2, \cdots$

NONINTERLACED Variables are numbered in the order of $u_1, u_2, \cdots, v_1, v_2, \cdots, p_1, p_2, \cdots$

3.2 parms_Mat object

The functions to create a parms_Mat object in pARMS are parms_MatCreate(&mat, map). This is much simpler compared to similar functions in other packages such as (PETSc [2] and Hypre [5]). Like PETSc and Hypre, pARMS insert entries with global indices into a matrix or vector object, which serves to alleviate users' programming burden. Unlike PETSc, pARMS assumes the entries needed by each processor reside in the processor locally. Thus, there are no communications involved when the matrix is assembled by calling parms_MatSetup. The motivation for this assumption is that an application programmer knows the set of vertices on each processor and therefore only assembles the local equations.

Vectors in this version of the pARMS package utilize C array pointers. This is done to simplify the code, and allow for ease when using the code in an application program.

3.3 parms_PC object

Currently, pARMS supplies three types of parallel preconditioners: block Jacobi (PCBJ), restricted additive Schwarz (PCRAS) [4], and Schur complement (PCSCHUR) [11, 8]. There are five types of local ILU preconditioners available in pARMS: ILU0(PCILU0), ILUK(PCILUK), ILUT(PCILUT), ARMS with symmetric permutation(PCARMS), and ARMS with nonsymmetric permutation(PCARMS) [10].

pARMS will include some new preconditioners in the next version, such as the restricted version of the overlapping Schur complement preconditioner (SchurRAS) [7], and inverse-based multilevel ILU preconditioner [3].

4 Template drivers in pARMS

There are a few template programs that may be compiled and run to solve a distributed linear system. These templates are provided in the examples/general, examples/grid, and examples/petsc directories, which are used to solve, respectively, (i) a general sparse linear system stored in the Harwell-Boeing sparse matrix (or user-defined) format as a single file; (ii) a model partial differential equation generated in a distributed manner, such that each processor generates its local points only; and (iii) PETSc version of solving a general sparse linear system stored in the Harwell-Boeing sparse matrix. All the template drivers use preconditioned (F)GMRES. A user may copy an example program as a starting point for developing their own application code.

Users may refer to 'README' in the directory for details about how to compile, link, and run driver programs. For example, in order to compile examples in the examples/general, the following steps are applied:

1. Edit makefile in the directory examples/general. Change the definitions of XIFLAGS and XLIB. XIFLAGS refers to the path of the header file of a graph partitioning software. XLIB refers to the path of the graph partitioning library.

2. Type 'make allexe' in directories examples/general and examples/grid to build all example programs.

References

[1] ITSOL. http://www-users.cs.umn.edu/ saad/software/ITSOL/index.html.

[2] Sattish Balay, William D. Gropp, Lois C. McInnes, and Barry F. Smith. PETSc home page. http://www.mcs.anl.gov/petsc.

[3] M. Bollhöfer and Y. Saad. Multilevel preconditioner constructed from inversed–based ilus. *SIAM Journal on Scientific Computing*, 27(5):1627–1650, 2006.

[4] X.-C. Cai and M. Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21:792–797, 1999.

[5] E. Chow, A. Cleary, and R. Falgout. *Hypre* User's manual, version 1.6.0. Technical Report UCRL-MA-137155, Lawrence Livermore National Laboratory, Livermore, CA, 1998.

[6] G. Karypis and V. Kumar. ParMETIS: Parallel graph partitioning and sparse matrix ordering library. Technical Report Tech.Rep. 97-060, University of Minnesota, 1997.

[7] Z. Li and Y. Saad. SchurRAS : A restricted version of the overlapping schur complement preconditioner. *SIAM Journal on Scientific Computing*, 27(5):1781–1801, 2006.

[8] Z. Li, Y. Saad, and M. Sosonkina. pARMS: A parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications*, 10:485–509, 2003.

[9] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2nd edition, 2003.

[10] Y. Saad. Multilevel ILU with reorderings for diagonal dominance. *SIAM Journal on Scientific Comuting*, 3(27):1032–1057, 2006.

[11] Y. Saad and M. Sosonkina. Distributed schur complement techniques for general sparse linear systems. *SIAM Journal on Scieific Computing*, 21(4):1337–1356, 1999.