Solving Linear Systems Using PARDISO

Ethan Metsger

1 Introduction

This document briefly describes some work I have done to simplify the use of PARDISO when attempting to solve linear systems. Tight integration with your software may be difficult to do, but the brief tutorial should also provide enough detail to get started. In particular, I hope that this permits a relatively easy means to abstract away a number of parameters used by PARDISO.

1.1 Where to get PARDISO

If you wish to obtain the PARDISO libraries, you can get them from http://www.computational.unibas.ch/cs/scicomp/software/pardiso/. You will have to fill out a download form in order to obtain a license to use the software. This license must be stored in a file called pardiso.lic in your home directory. If you would rather not download the libraries, you can use the ones I have downloaded; they are in my public repository: /home/1/metsger/pub/lib/mat/pardiso. Regardless of how you obtain the libraries, you *must* obtain a license code for your own username and machine.

Your license is good for one hostname and username, e.g., omicron and metsger. Do not put the fully qualified domain name (e.g. omicron.cse.ohio-state.edu) in the download form. Otherwise, you will get a nasty error message when attempting to run software compiled with the PARDISO library. You will not be able to run programs on a different hostname, either.

1.2 Other Libraries

You may find that other libraries will be useful and/or needed when you use PARDISO. In particular, the ATLAS libraries may be needed. You can link against the ATLAS libraries by referencing them from my public repository: /home/1/metsger/pub/lib/mat.

1.3 Supported Matrix Types

PARDISO supports many types of matrices. These are listed in a table on page 7 of the user guide, which you may find online. It is my understanding that solving a linear system using the Conjugate Gradient Method (as would be used for implicit Euler) requires a symmetric positive-definite matrix, so this matrix type is assumed in the program.

2 The Simplified PARDISO

The University of Basel has put up several different examples of how you may be able to use PARDISO. I have expanded on their example by including a small function that reads sparse matrix data from a file. This should allow relatively transparent matrix solving.

2.1 Where to Find

My implementation of the functions described below may be found at

`/home/1/metsger /pub/888x14/matrix/prog/`

2.2 Sparse Array Representation

A sparse array is specified by three different arrays:

1. A packed array (no zeros) containing the data.
2. An array of the same size in 1 which contains the column location of each element in the array.
3. An array which tells which index in the array of 1 starts the next row. This always has one more than the number of rows in the array. Essentially, this specifies where each row starts in the packed array.

This information may be found on page five of the PARDISO user guide (version 1.2.3) available from the website.

Please note that we expect the arrays to be in C-style format; that is, arrays begin at element zero and continue to `length - 1`. PARDISO is written in FORTRAN, and so has somewhat different conventions, but these are accommodated in the implementation.

2.3 Function API

There are three functions associated with this simplifying library. The first essentially encapsulates an example program provided by the programmers of PARDISO. The second permits the reading of an array from a file, and the third writes the data to a file. These two functions are provided so that you can simply plug them if needed.

The file manipulation commands are not terribly useful, I don't think. However, they might find some utility while doing unit tests of your code or attempting to store intermediate information about the system you are attempting to solve.

`float *x14_solve(double *a, int *ja, int *ia, int n)` This solves the linear equation specified by the three arrays given as parameters.

`a` is the packed array that holds the sparse matrix.

`ja` is the array that holds the column locations for the elements of `a`.

`ia` holds the row locations as specified above. The size of `ia` is one more than the number of rows; the last element of the array should be the length of `ja + 1`.

`n` is the number of rows in the sparse matrix.

void x14_read(char *filename, double **a, int **ja, int **ia, int *n) This reads array data from the filename specified as `filename`. As an example, you should call it like this:

```
char *f = "data.txt";
double *a;
int *ja, *ia, n;

x14_read (f, &a, &ja, &ia, &n);
```

Unfortunately, we have to use somewhat obfuscated syntax because of the pass-by-value semantics of the language.

void x14_write(char *filename, double *a, int *ja, int *ia) This writes the data to a file. **THIS IS CURRENTLY UNIMPLEMENTED.**

2.4 File Format

The file format is as follows:

```
<number of elements in packed array>
<element 1>
<element 2>
                                    ⋮
<element n>
<number of elements in column array>
<element 1>
<element 2>
                                    ⋮
<element m>
<number of elements in row array>
<element 1>
<element 2>
                                    ⋮
<element o>
```

2.5 Example Code Snippet

Here is a quick example of how you might load matrix data from a file and then solve it:

```
#include <stdio.h>

extern x14_read (char *, double **, int **, int **, int *);
extern x14_solve (double *, int *, int *, int);
```

```
int main (int argc, char ** argv)
{
  double *a;
  int *ja, *ia, n;

  x14_read (argv[1], &a, &ja, &ia, &n);
  double *d = x14_solve (a, ja, ia, n);

  int i;
  for (i = 0; i < n; i ++) {
    printf ("%f\r\n", d[i]);
  }
}
```

3 Using the PARDISO Libraries

3.1 Linking

When you link against the PARDISO libraries, you will need to compile using a command something like this:

```
gcc -g -o <exeName> <sourceName> -L/home/1/metsger/pub/lib/mat/ \
    -L/home/1/metsger/pub/lib/mat/pardiso/ \
    -lpardiso_SUN32 -lm x14.o
```

While adding the debugging information (the -g flag) isn't strictly necessary (and may cause some extra bloat), it can be helpful when you're attempting to find out why your program segfaulted.

In order to use the methods described above, you need to link against `x14.o` as well.

3.2 Running

You will find that running programs linked against PARDISO libraries may be difficult because the libraries are not in `ld`'s search path. To change this, you should do something like this in `bash` :

```
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}: \
                       /home/1/metsger/pub/lib/mat/: \
                       /home/1/metsger/lib/mat/pardiso

export OMP_NUM_THREADS=1
```

This will set the libraries in the search path as well as specifying that you are using a single processor. Without the second line, you will have an error when running the program.

I created a small shell script that would set these environment variables for me and then execute the program, which seemed to do fairly well.