Artifact for paper 163 of Euro-Par 2020, ”Improving mapping for sparse direct solvers: A trade-off between data locality and load balancing” by authors Changjiang Gou, Ali Al Zoobi, Anne Benoit, Mathieu Faverge, Loris Marchal, Grgoire Pichon, and Pierre Ramet.

Please note that a `README.md` file is provided to ease the process and copy/paste commands.

1 Getting Started Guide

To generate the results of the paper, one needs to install the PaStiX software and the R script language.

1.1 Required softwares

PaStiX. PaStiX (Parallel Sparse matriX package) is a scientific library that provides a high performance parallel solver for very large sparse linear systems based on direct methods. The introduction and installation steps can be found here. The PaStiX version used for our experiments is a modified 6.1.0 release tagged as `europar2020` on the public git repository.

R. R is a language and environment for statistical computing and graphics. We use it to gather all raw data generated by PaStiX and to plot figures. R version 3.3.2 (2016-10-31) is used in our experiment. Introduction and installation steps can be found here.

We also provided a script file, *env-pastix.sh*, that lists all the dependencies we used on the experiment platform with their exact version numbers.

1.2 Instructions to install PaStiX

We consider in the following that the user has setup a `ARTIFACTS_DIR` variable to define the directory where the artifacts have been unfolded.

1. install library dependencies: for instance, on debian-like systems, the required dependencies can be installed with the following command:

```
sudo apt-get install git cmake g++ gfortran \
                     libhwloc-dev libscotch-dev \
                     libopenblas-dev liblapacke-dev \
                     python-numpy openmpi-bin
```

2. Go to the artifacts directory and clone the repository:

```
cd $ARTIFACTS_DIR
git clone -b europar2020 --recursive \
          https://gitlab.inria.fr/solverstack/pastix.git
```

At this point, the PaStiX solver should have been properly downloaded. For the ease of reproducing all results, we propose to build two versions of the PaStiX library. One without the MPI support installed in the directory `$ARTIFACTS_DIR/pastix_sharedM`, and one with MPI support installed in the directory `$ARTIFACTS_DIR/pastix_distributed`. The next section summarizes how to relaunch experiments and obtain the figures of the paper.

Let’s first create the shared memory build, and test it:

```
cd $ARTIFACTS_DIR/pastix_sharedM
cmake $ARTIFACTS_DIR/pastix -DPASTIX_WITH_MPI=OFF -DPASTIX_INT64=OFF
make -j 8
./example/simple -9 20:20:20
```

Let’s now create the distributed memory build, and test it:

```
cd $ARTIFACTS_DIR/pastix_distributed
cmake $ARTIFACTS_DIR/pastix -DPASTIX_WITH_MPI=ON -DPASTIX_INT64=OFF
make -j 8
mpirun -np 2 ./example/simple -9 20:20:20
```

Note that for the experiments we used `-DPASTIX_INT64=ON`, as we used our own version of Scotch compiled with 64 bits integer support.

2 Step-by-Step Instructions to reproduce the results

The platform we used is equipped with two Intel Xeon E5-2680v3 12-cores running at 2.50 GHz and 128 GB of memory. Another shared memory experiment was performed on a node equipped with four Intel Xeon E5-4620 8-cores running at 2.20 GHz and 378 GB of memory. The whole execution takes more than one day. Getting all results on one matrix takes 1.5 hours averagely.

We provided a script file *env-pastix.sh* to load all the dependencies used on the platform. To know exactly which version of the dependencies we used, look at the file. To load these dependencies, we used: *source env-pastix.sh*. Note that this file is specific to our platform, and is not meant to be used on any other architecture.

2.1 Download matrices

You can find 33 out of the 34 matrices used in our paper on the SuiteSparse Matrix Collection website: SuiteSparse Matrix Collection. We provide a script to download all those 33 matrices: **download_matrices.sh**. The remaining matrix, `matr5`, is unfortunately unavailable publicly. We suppose that all matrices are in directory **matrice** of the **artifact** directory.

2.2 Run PaStiX on an example matrix

We provide a small example matrix **1138_bus.mtx** in the **matrice** directory to show how to generate each figure's raw data in a short time (less than 3 minutes).

Get the simulation cost, estimated factorization time and data movement matrix on shared memory environment

- go to the execution directory:

  ```
  cd $ARTIFACTS_DIR/pastix\_sharedM/example
  ```

- run PaStiX with heuristic PropMap:

  ```
  ./analyze -3 $ARTIFACTS_DIR/matrice/1138_bus.mtx -v4 -i iparm_allcand 0
  ```

  On the output of the screen, from which an example is given here:

  ```
    Analyse step:
  OUTPUTDIR: pastix-jBBJqX
      ...
      Simulation done in                    1.974106e-04 s
      ...
      Prediction:
        Model                              AMD 6180  MKL
        Time to factorize                   1.572448e-03 s
      Time for analyze                      4.004240e-03 s
  ```

  Time to factorize in the *Analyse step* section indicates the estimated factorization time, *simulation done in* indicates the simulation time cost, *OUTPUTDIR: pastix-** indicates where the csv file is. It stores the data movements inside an MPI node. Results generated correspond to figure 4 and 5.

- the option *-i iparm_allcand* with values 0, 1, 3, 4, can get respectively the results for heuristic PropMap, All2All, Steal, StealLocal.

Get the simulation cost, estimated factorization time and data movement matrix on distributed memory environment

- Run PaStiX on, let's say, 6 MPI nodes, each with 4 threads:

  ```
  mpirun -np 6 $ARTIFACTS_DIR/pastix_distributed/example/analyze \
         -3 $ARTIFACTS_DIR/matrice/1138_bus.mtx \
         -v4 -t 4 -i iparm_allcand 0
  ```

  With options *-np* (of mpirun), and *t* of PaStiX, we can set the number of MPI nodes and threads respectively. Results generated corresponds to figure 3 and 5.

- It will generate similar results as in section 2.2.
- the heuristic can also be set with option *-i iparm_allcand*.

Get the factorization time We now use the `bench_facto` binary to get the factorization time.

```
$ARTIFACTS_DIR/pastix_sharedM/example/bench_facto \
    -3 $ARTIFACTS_DIR/matrice/1138_bus.mtx -v4 -i iparm_allcand 0
```

- On the output, *Time to factorize* in the *Factorization step* indicates the factorization time. We have three factorization rounds to get an average.
- same as before, you can select the heuristic by flag *-i iparm_allcand*. Results generated corresponds to figure 6.

2.3 Run scripts to generate results

We provided several bash scripts in the SCRIPTS directory to run PaStiX on the 34 matrices. The whole execution takes one or two days. So we also provide the raw results. You can start from the raw results to generate figures of our paper, see section 2.4.

2.3.1 Generate results on shared memory environment

basic_forloop_sharedM.sh and **gen_all_sharedM.sh** are the scripts to do all analysis steps on shared memory environment.

On the platform we have used for the experiments, it is done using Slurm job scheduler with the command line:

```
sbatch ./gen_all_sharedM.sh
```

The results generated correspond to figure 4 and 5 in our paper.

2.3.2 Generate analysis results on distributed environment

basic_forloop_distributed.sh and **gen_all_distributed.sh** are scripts to do analysis steps on distributed environment. One needs to set the right number of MPI nodes and threads each MPI node has.

- on line 15 and 16 of file **basic_forloop_distributed.sh**, set the number of threads and number of MPI nodes respectively. For example, we set *NBCORES=6* and *NPROC=4* if we want have the result on "4M6".

Then run the bash script by *sbatch ./gen_all_distributed.sh* to get the results. The results generated correspond to figure 3 and 5 in our paper.

2.3.3 Generate factorization results on shared memory environment

This subsection will factorize all matrices to obtain the factorization time, which corresponds to raw data in Fig.6 of our paper. **run_one_matrix_factor.sh** script can be used to obtain the factorization times for one matrix, while **run_all_matrix_factor.sh** launches all the experiments.

You can either submit a single matrix factorization with:

```
sbatch ./run_one_matrix_factor.sh nd24k -f 0 --mm ../matrice/nd24k.mtx
```

or submit all matrices factorization with:

```
sbatch ./run_all_matrix_factor.sh
```

2.4 Generate plots from raw results

We provided the raw results generated from section 2.3. From here, you can generate all plots in our paper, it takes less than 10 minutes. Please make sure *R* is installed first. Then, you just need to go to the directory **raw_results** and run command *Rscript CombineData_Plot.R*. A pdf file named Rplots will be generated.