Recursive Algebraic Coloring Engine

Artifact description

Christie Louis Alappat
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
christie.alappat@fau.de

Georg Hager
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
georg.hager@fau.de

Gerhard Wellein
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
gerhard.wellein@fau.de

Olaf Schenk
Institute of Computational Science
Lugano, Switzerland
olaf.schenk@usi.ch

A Abstract

This article describes the experiment settings and how the performance runs were conducted for the results shown in the SC18 poster "Recursive Algebriac Coloring Engine". It gives a detailed information on how to get, compile and perform the runs using RACE library. Furthermore it also gives a brief description on the softwares and method used to perform the comparison runs.

B Description

B.1 Check-list

- **Algorithm**: RAC method explained in poster and extended summary.
- **Program:** C++,C program and libraries
- **Compilation:** Intel compiler v 17.0.3 with -O3 -xHOST
- **Hardware:** SkyLake (Gold 6148) @ 2.4 GHz and Ivy Bridge E5-2660 v2 @ 2.2 GHz
- **Run-time environment:** Ubuntu 16.04.5 LTS on SkyLake and CentOS Linux release 7.5.1804 on Ivy Bridge.
- **Data set:** Publicly available matrices.
- **Output:** performance in GFlop/s and iterations.
- **Experiment workflow:** Clone and install the RACE-AD repository, build, compile, run.
- **Publicly available:** Pre-compiled version of the RACE library is publicly available.

B.2 How software can be obtained

Clone the git repository from the URL:https://bitbucket.org/christiealappatt/race-ad/src/master/. It contains pre-compiled version of the RACE library and example kernels required for the sc18 poster. The main source code of the RACE library can be found under https://bitbucket.org/christiealappatt/race/src/master/ which is expected to be made public by October 2018.

B.3 Hardware dependencies

Multicore processors. Currently RACE library has been tested only for Intel and AMD systems.

B.4 Software dependencies

RACE requires hwloc [1] library. If not found hwloc will be downloaded and installed. If RCM is used in the level construction phase (step 1 of RAC method) Intel SpMP [8] is also required. The library will also be downloaded and installed automatically in build process.

B.5 Datasets

24 of the matrices are from well-known SuiteSparse Matrix collection [2] (former University of Florida Sparse Matrix collection). Rest 4 matrices come from quantum physics field and can be generated using GHOST and PHYSICS library [3] which is publicly available.

C Installation

1. Clone the git repository.

```
$ git clone\
  git@bitbucket.org:christiealappatt/race-ad.git
$ cd race-ad
```

2. Configure the build using cmake. Set RACE_USE_SPMP for using RCM instead of BFS in level construction phase. This is done for experiment runs shown in poster.

```
$ mkdir build && cd build
$ CC=icc CXX=icpc cmake .. -DRACE_USE_SPMP=ON
```

3. Compile two provided executables `race` and `symm_kacz_convergence`. If dependencies are not installed this step will download and install them. Please ensure internet connectivity during the first make.

```
$ make
```

SC'18, November 2018, Dallas,TX USA
2018.

D Experiment workflow

To run the performance benchmarks of SymmSpMV and SymmKACZ shown in the poster use the `race` executable.

To do the convergence analysis of Symmetric Kaczmarz (SymmKACZ) use `symm_kacz_convergence` executable. The workflow of the convergence test is as follows

1. Measure the L2 norm error reached by serial (1 thread) run for 1000 iterations. This is fixed as baseline.
2. Now do the parallel run and record the iterations required to reach the same error as baseline.

For comparison with Multicoloring (MC) we used COLPACK [4] library to color (pre-processing step) the matrix. In case of Algebraic Block Multicoloring (ABMC) we used first METIS [7] to partition the matrix into blocks and then we used COLPACK to color these blocks. A tuning was done from small (4) to large block size (128) as shown in [6]. Similar kernels to that of RACE was run with these methods. For comparison of SymmSpMV with MKL[5] we used `mkl_cspblas_dcsrsymv()` routine. The test scenario and work flow was identical to that of RACE runs.

To do hardware performance counter measurements we used LIKWID [9]. This has been used to generate the plot of memory data traffic used in the poster.

E Evaluation and expected result

All runs shown in the poster were conducted with pinned threads and fixed core and uncore clock speed (2.4 GHz SkyLake and 2.2 GHz IvyBridge). While executing kernels with RACE it automatically pins the threads, but for kernels written outside RACE we need to pin using tools like `likwid-pin`. Since the measurements are done only on one socket its also sufficient to use `taskset`.

Pre-processing time were not included for any of the shown results. An efficiency (parameter) of 80% (constant value) was fixed for each recursive stage of the RACE preprocessing.

The performance results reported were mean of 1000 iterations. The result can be obtained by running the following command:

```
$ OMP_NUM_THREADS=$nthreads RACE_EFFICIENCY=80,80\
 taskset -c 0-$((nthreads-1)) ./race \
 -m [matrix file] -c $nthreads -i 1000
```

During the run it displays the RAC tree structure created for the specific matrix. Finally it reports the performance in GFlop/s of different kernels like sparse matrix transpose vector, symmetric sparse matrix vector, kaczmarz, gauss-seidel etc. run with RACE. Performance of SpMV kernel is also reported for comparison.

```
SPMV : 16.4880 GFlop/s ; Time = 0.49739 s
SPMTV : 16.4264 GFlop/s ; Time = 0.49926 s
KACZ : 30.1878 GFlop/s ; Time = 0.54333 s
SYMM_KACZ : 27.9842 GFlop/s ; Time = 1.17224 s
GS : 16.4741 GFlop/s ; Time = 0.49781 s
SYMM_SPMV : 30.4742 GFlop/s ; Time = 0.26911 s
```

For obtaining the convergence results of symmKACZ one has to specify the maximum number of iterations and desired L2 error to be achieved. The run is as follows:

```
$ OMP_NUM_THREADS=$nthreads RACE_EFFICIENCY=80,80\
 taskset -c 0-$((nthreads-1)) ./symm_kacz_convergence\
  -m [matrix] -c $nthreads -i [max. iter] -T [error]
```

The run would give back the performance, actual error and iterations achieved after the run.

References

[1] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst. 2010. hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. 180–186. https://doi.org/10.1109/PDP.2010.67

[2] Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. https://sparse.tamu.edu/about. *ACM Trans. Math. Softw.* 38, 1, Article 1 (Dec. 2011), 25 pages. https://doi.org/10.1145/2049662.2049663

[3] ESSEX project. -. GHOST and PHYSICS library. https://bitbucket.org/essex/

[4] Assefaw H. Gebremedhin, Duc Nguyen, Md. Mostofa Ali Patwary, and Alex Pothen. 2013. ColPack: Software for Graph Coloring and Related Problems in Scientific Computing. *ACM Trans. Math. Softw.* 40, 1, Article 1 (Oct. 2013), 31 pages. https://doi.org/10.1145/2513109.2513110

[5] Intel. [n. d.]. Intel Math Kernel Library. https://software.intel.com/en-us/mkl

[6] Takeshi Iwashita, Hiroshi Nakashima, and Yasuhito Takahashi. 2012. Algebraic Block Multi-Color Ordering Method for Parallel Multi-Threaded Sparse Triangular Solver in ICCG Method. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS '12)*. IEEE Computer Society, Washington, DC, USA, 474–483. https://doi.org/10.1109/IPDPS.2012.51

[7] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing* 20, 1 (1998), 359–392. https://doi.org/10.1137/S1064827595287997

[8] SpMP Development Team. -. Sparse matrix pre-processing library. https://github.com/IntelLabs/SpMP

[9] J. Treibig, G. Hager, and G. Wellein. 2010. LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments. (2010).