LU Factorization with Partial Pivoting with OpenMP Tasks

Yu Pei

May 3rd, 2017

COSC594 Final Presentation

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Outline

1. Objective
2. Background
    1. LU Factorization with Partial Pivoting
    2. OpenMP Task Pragma
3. Implementation
4. Experiment Results
5. Conclusion and Future Works

Objective

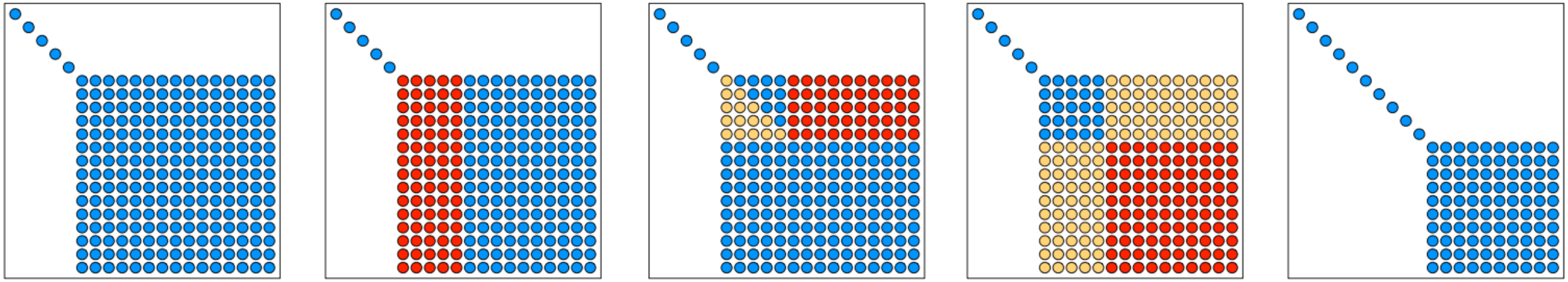
LU factorization is a widely used algorithm to calculate matrix inverse, determinant etc.

It is the core computation of many applications, thus has been a prime target for aggressive optimizations.

For shared memory system, LAPACK provides the classical solution based on block data layout (BLAS 3) and bulk synchronous parallelism (BSP).

With runtime scheduling of tasks, we can utilize the underlying heterogeneous system more efficiently and achieve better performance (PLASMA library with this purpose).

Background – LU with Partial Pivoting



(From Mark Gates Lecture note)

Factor panel of n_b columns

- getrf2, unblocked BLAS-2 code

Level 3 BLAS update block-row of U

- trsm

Level 3 BLAS update trailing matrix

- gemm
- Aimed at machines with cache hierachy

Bulk synchronous

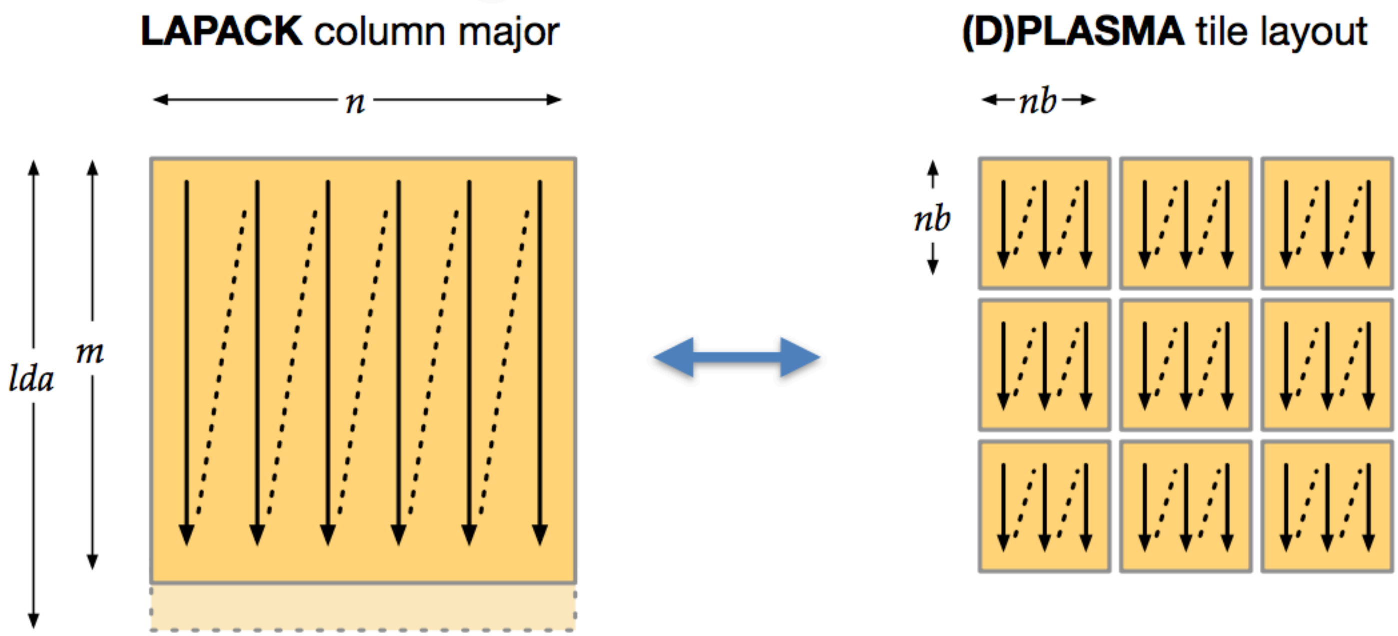
Background – Dynamic scheduling runtimes

Cilk	MIT
Jade	Stanford University
SMPSs / OMPSs	Barcelona Supercomputer Center
StarPU	INRIA Bordeaux
QUARK	University of Tennessee
SuperGlue & DuctTeiP	Uppsala University

OpenMP 4

	OpenMP	GNU	
May 2008	OpenMP 3.0		#pragma omp task
April 2009		GCC 4.4	
July 2013	OpenMP 4.0		#pragma omp task **depend**
April 2014		GCC 4.9	

Implementation – Tile layout



Each tile is contiguous (column major)

Enables dataflow scheduling

Cache and TLB efficient (reduces conflict misses and false sharing)

In-place, parallel layout translation

Implementation – Pseudocode

```
column_to_tile(pA, A, NB); // Convert to tile layout

#pragma omp parallel
#pragma omp master
{
  for(k=0; k<num_tiles; k++){ // Loop over columns
    #pragma omp task depend(inout: A(0, k)[M*NB]) \
                     depend(out: ipiv[k*NB:NB]
    {
      dgetrf(); // Panel factorization
    }

    // update trailing submatrix
    for(j=k+1; j<num_tiles; j++){
      #pragma omp task depend(in: A(0, k)[M*NB])    \
                       depend(in: ipiv[k*NB:NB])    \
                       depend(inout: A(0, j)[M*NB]) \
      {
        core_geswp(); // line swaps
        dtrsm();      // triangular solve
        dgemm();      // schur's complement
      }
    }

  }
  // pivoting to the left
  for(t=1; t<num_tiles; t++){
    #pragma omp task depend(in: ipiv[(nt-1)*NB:NB])  \
                     depend(inout: A(0, t)[M*NB])
    {
      core_geswp();
    }
  }
}

tile_to_column(pA, A, NB); // Convert back to column major layout
```

Master thread create tasks.

Three set of tasks as directed acyclic graph (DAG).

Coarser dependency specification than in PLASMA.

Merging of three operations within one task (swaps, trsm and gemm).

Experiments

Experiments on a machine 12 Intel Haswell threads on each socket (Xeon E5-2680 v3 2.60 GHz) for a total of 24 threads.

The peak double precision performance is 960 GFlop/s.

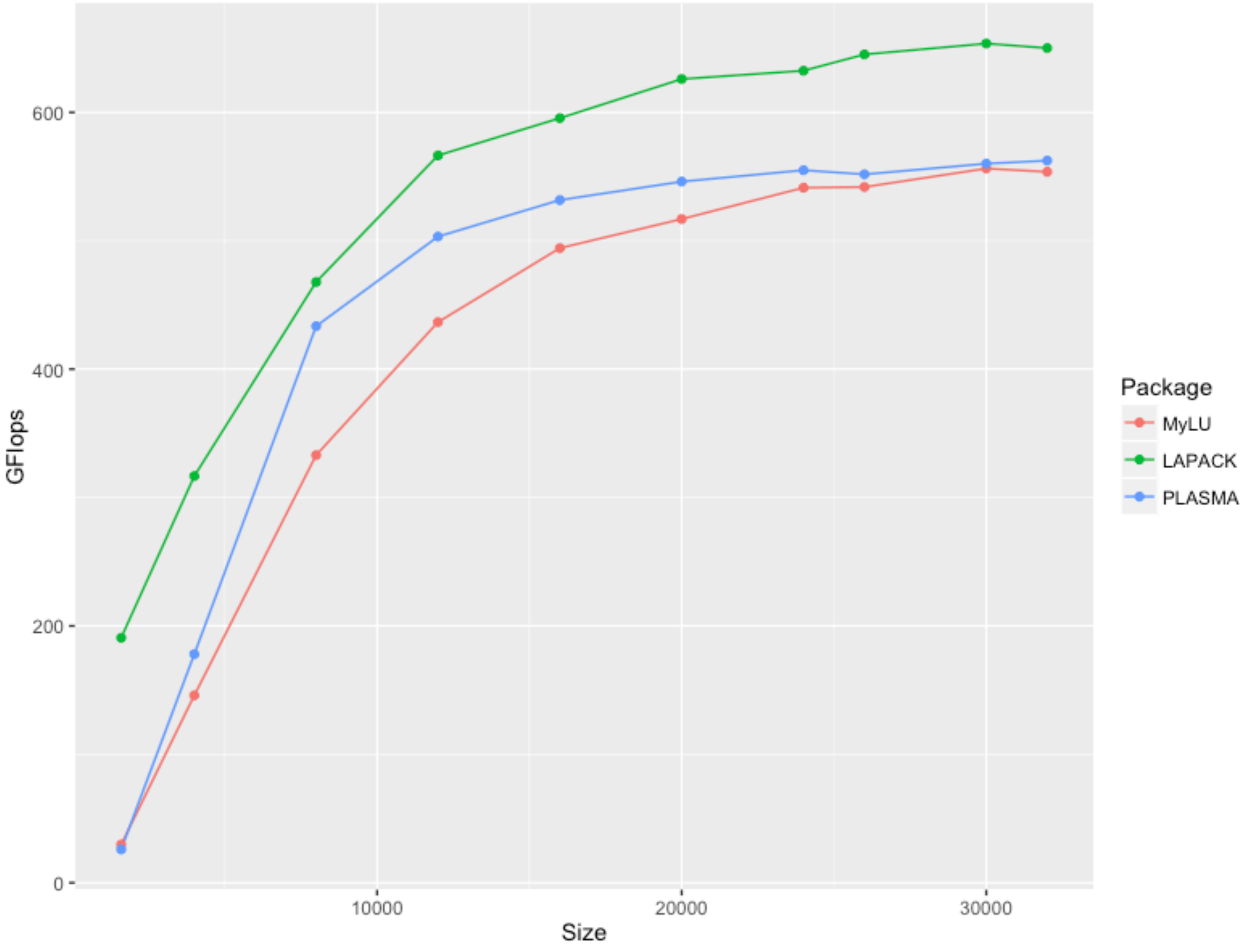
ICC compiler 16.0.3 and the corresponding OpenMP and MKL math library were used for optimized BLAS operations.

Result validated with MKL LAPACK implementation.

Tile size is fixed to be 200 in our code as well as in PLASMA, and PLASMA panel block size is set to 40, with 5 parallel threads.

Results

Compared with PLASMA and MKL LAPACK implementation, varying the matrix size.



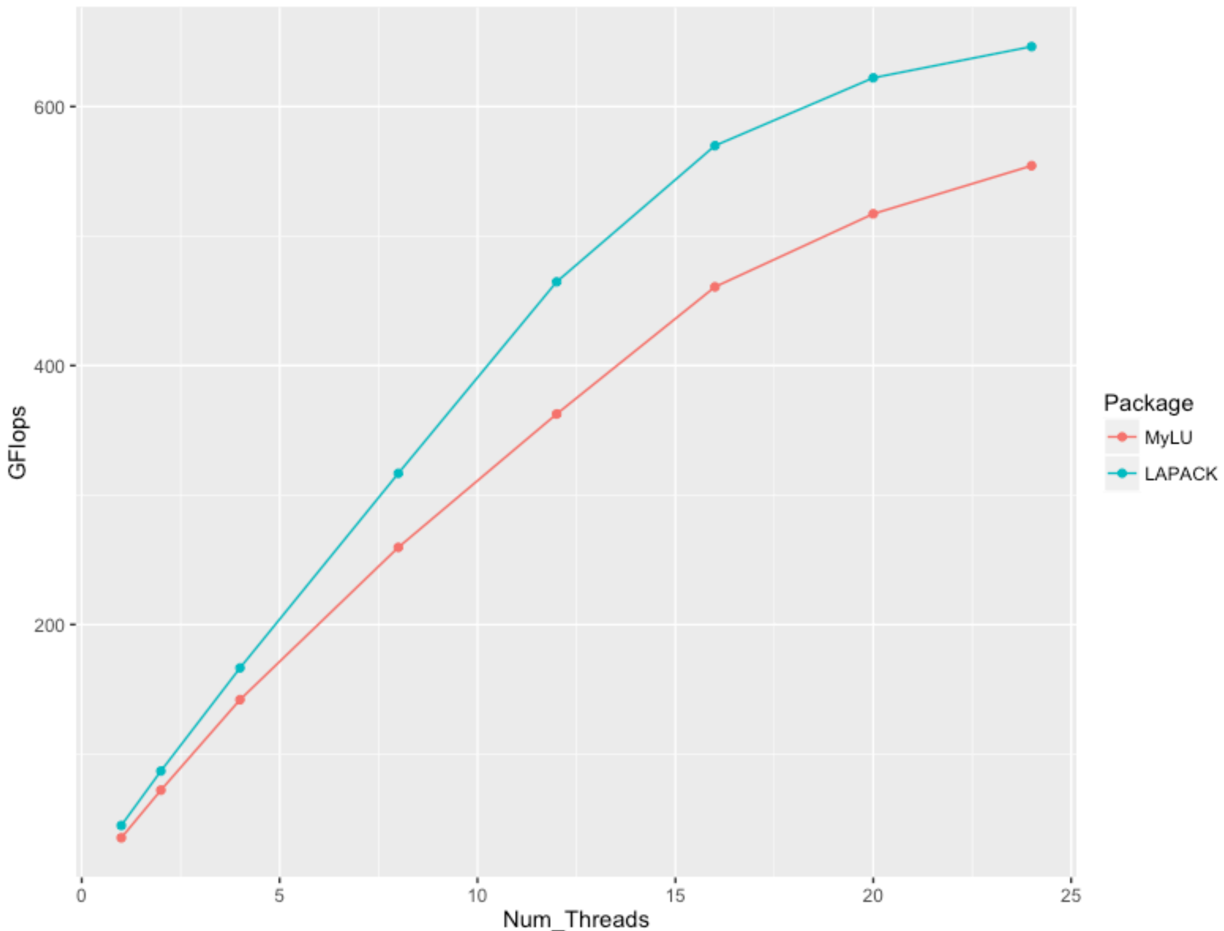
Timing of my implementation excludes conversion between the two layouts.

MKL is faster than our implementation

PLASMA saturates faster than my code, likely due to parallel panel factorization.

Results

Next we focus on the scalability of tasks LU regarding the number of thread used.



Conclusions

In this project, we experienced with task scheduling, specifically the OpenMP task framework and used it to implement the classical LU matrix factorization.

There were some issues regarding dependency specification during the implementation and we solved the problem by expanding and unifying the pointers starting position.

Comparable performance results are obtained for our implementation when compared with PLASMA, although still fall behind MKL LAPACK implementation.

Future Works

- Parallel panel factorization. Since we know that panel factorization is in the critical path and introducing BLAS 3 operations and multiple threads will yield sizable speedup. PLASMA has the routine that implements this strategy.

- Speedup with accelerators. A large portion of the work lies in the matrix matrix multiplication and this can be significantly accelerated with GPU for example. Incorporating GPU can achieve much higher computing power but synchronization between host and device needs to be carefully managed.

- Assigning task priority. OpenMP provides a way to hint to the runtime the priority of the tasks. Giving tasks on the critical path should be able to improve the performance, and tracing graph can help in that aspect.

References

1. Jack Dongarra, Mathieu Faverge, Hatem Ltaief and Piotr Luszcsek. *Achieving Numerical Accuracy and High Performance using Recursive Tile LU Factorization.* LAPACK Working Note 259.
2. Jakub Kurzak, Piotr Luszczek, Mathieu Faverge and Jack Dongarra. *LU Factorization with Partial Pivoting for a Multi-CPU, Multi-GPU Shared Memory System.* LAPACK Working Note 266.
3. Asim YarKhan, Jakub Kurzak, Piotr Luszczek and Jack Dongarra. *Porting the PLASMA Numerical Library to the OpenMP Standard.* Int J Parallel Prog (2017) 45: 612. doi:10.1007/s10766-016-0441-6.
4. Duran, A., Ayguadé, E., Badia, R.M., Labarta, J., Martinell, L., Martorell, X., Planas, J. *OMPSS: a proposal for programming heterogeneous multi-core architectures.* Parallel Process. Lett. 21(02),173–193 (2011)
5. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.-A. *StarPU: a unified platform for task schedulingon heterogeneous multicore architectures.* Concurr. Comput.: Pract. Exp. 23(2), 187–198 (2011)
6. YarKhan, A. *Dynamic Task Execution on Shared and Distributed Memory Architectures.* PhD thesis, University of Tennessee (2012)
7. Drebes A., Bréjon JB., Pop A., Heydemann K., Cohen A. *Language-Centric Performance Analysis of OpenMP Programs with Aftermath.* (2016) In: Maruyama N., de Supinski B., Wahib M. (eds) OpenMP: Memory, Devices, and Tasks. IWOMP 2016. Lecture Notes in Computer Science, vol 9903. Springer, Cham

Questions?

Thank You