OpenMP Tasks and PLASMA

Asim YarKhan

ICL Lunch Talk

Sep 11 2015

INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

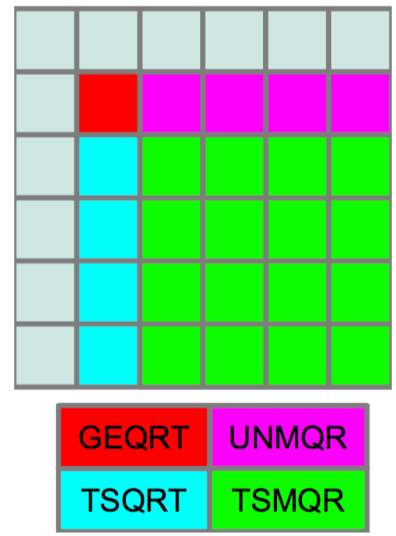
Preamble and Question

- Tiled linear algebra --- PLASMA
  - Async tasks, better cache use
- Superscalar execution is nice
  - Hard to write compact dags containing all explicit dependencies
  - Serial code leads to “correct” parallel execution
  - Possible bottleneck? Unrolling tasks/dependencies, hidden by task grain
- Runtime required – QUARK [or ...]
  - Nothing else appropriate available when we started
- Question
  - Now that OpenMP 4 has tasks with dependencies, is it useful/needful to maintain our own runtime environment?
  - Performance. Multithreaded tasks. Cancellation. Complex merged algorithms. Task priorities. Upper/lower triangle.

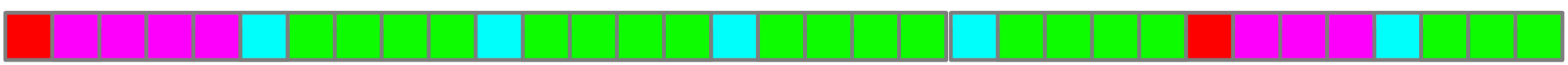
Superscalar Execution

- Tile QR factorization
  - Unroll a serial sequence of functions/tasks
  - Execute them respecting data hazards (RAW, WAR, WAW)

for k = 0 ... TILES−1
  gegrt(A_{kk}^{rw} , T_{kk}^{w})
  for n = k+1..TILES−1
    unmqr(A_{kk-low}^{r} , T_{kk}^{r} , A_{kn}^{rw})
  for m = k+1..TILES−1
    tsqrt(A_{kk-up}^{rw} , A_{mk}^{rw} , T_{mk}^{rw})
    for n = k+1..TILES−1
      tsmqr(A_{mk}^{r} , T_{mk}^{r} , A_{kn}^{rw} , A_{mn}^{rw})

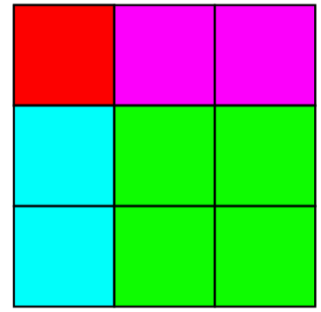


List of tasks as they are generated by the loops



Superscalar Execution

F0	geqrt(	A^{rw}_{00},	T^{w}_{00}			)
F1	unmqr(	A^{r}_{00},	T^{r}_{00},	A^{rw}_{01}		)
F2	unmqr(	A^{r}_{00},	T^{r}_{00},	A^{rw}_{02}		)
F3	tsqrt(	A^{rw}_{00},	A^{rw}_{10},	T^{w}_{10}		)
F4	tsmqr(	A^{rw}_{01},	A^{rw}_{11},	A^{r}_{10},	T^{r}_{10}	)
F5	tsmqr(	A^{rw}_{02},	A^{rw}_{12},	A^{r}_{10},	T^{r}_{10}	)
F6	tsqrt(	A^{rw}_{00},	A^{rw}_{20},	T^{w}_{20}		)
F7	tsmqr(	A^{rw}_{01},	A^{rw}_{21},	A^{r}_{20},	T^{r}_{20}	)
F8	tsmqr(	A^{rw}_{02},	A^{rw}_{22},	A^{r}_{20},	T^{r}_{20}	)
F9	geqrt(	A^{rw}_{11},	T^{w}_{11}			)
F10	unmqr(	A^{r}_{11},	T^{w}_{11},	A^{rw}_{12}		)
F11	tsqrt(	A^{rw}_{11},	A^{rw}_{21},	T^{w}_{21}		)
F12	tsmqr(	A^{rw}_{12},	A^{rw}_{22},	A^{r}_{21},	T^{r}_{21}	)
F13	geqrt(	A^{rw}_{22},	T^{w}_{22}			)



Data dependencies from the first five tasks in the QR factorization

A_{00} : $F0^{rw} : F1^{r} : F2^{r} : F3^{rw}$

A_{01} : $F1^{rw} : F4^{rw}$

A_{02} : $F2^{rw} : F5^{rw}$

A_{10} : $F3^{rw} : F4^{r} : F5^{r}$

A_{11} : $F4^{rw}$

A_{12} : $F5^{rw}$

A_{20} :

A_{21} :

A_{22} :

OpenMP

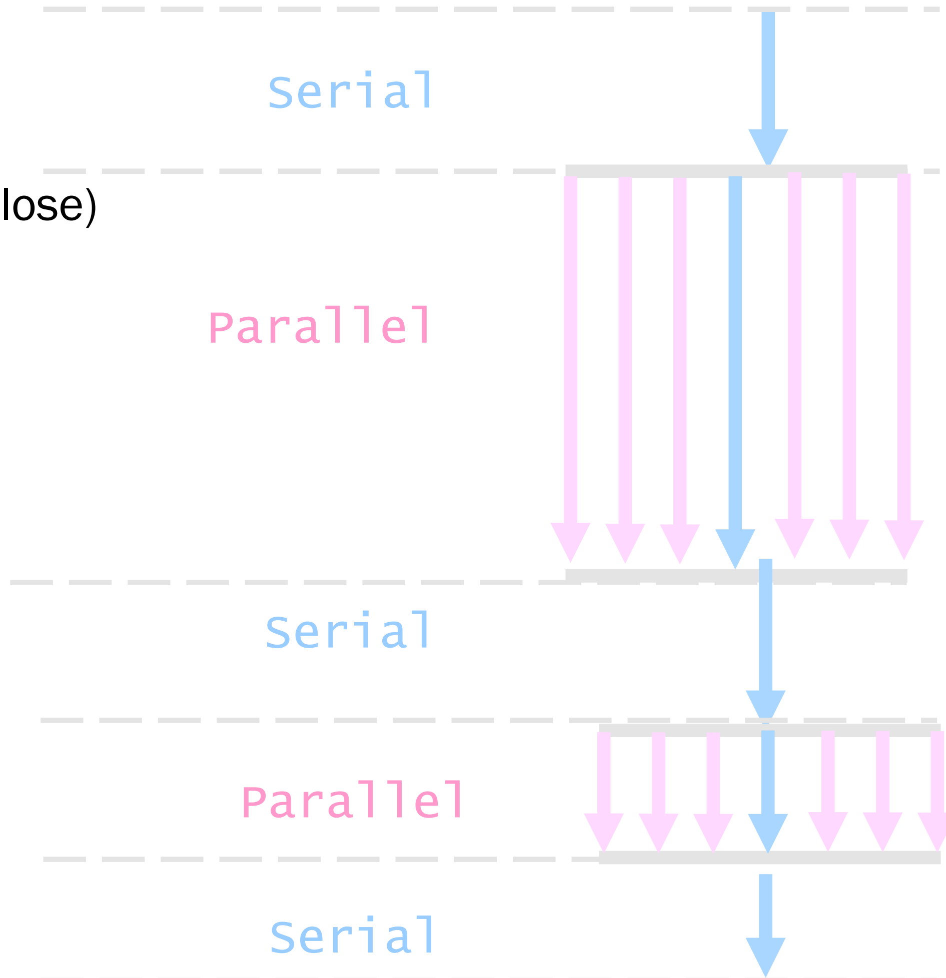
- Set of compiler directives, library routines, runtime for multi-threaded applications

```
#pragma omp parallel for
for(int x=0; x < width; x++)
    for(int y=0; y < height; y++)
        finalImage[x][y] = RenderPixel(x,y, &sceneData);
```

- Possible implementation
  - Creates a function for the “for” loop, passing in the visible variables as (shared, private, firstprivate, …). The loop parameter is mapped to a function parameter. The function is called by different threads using different loop parameters based on thread number.

OpenMP

```
#include <omp.h>
void main()
{
    #pragma omp parallel proc_bind(close)
    {
        int id = omp_get_thread_num()
        printf("%d\n", id );
        #pragma omp for
            for (int i=0; i<100; i++)
                dosomething(i);
    }
    compute_checks(something);
    #pragma omp parallel for
        for (int k=0; k<3; k++)
                A[i] = B[i] + C[i];
    print_result(A);
}
```



Variables (private, firstprivate, shared, ...) Reductions Schedule (static, dynamic [chunk])
Task Environment (NUM_THREADS, PROC_BIND, PLACES, WAIT_POLICY,...)

OpenMP Tasks with depend

- OpenMP 4.0
  - Standard released mid 2013
  - Added "depend" clause to tasks
- Working view
  - Create a function for the structured block
  - Pass in the visible variables (firstprivate, shared, …)
  - Run this function using threads when depends are satisfied

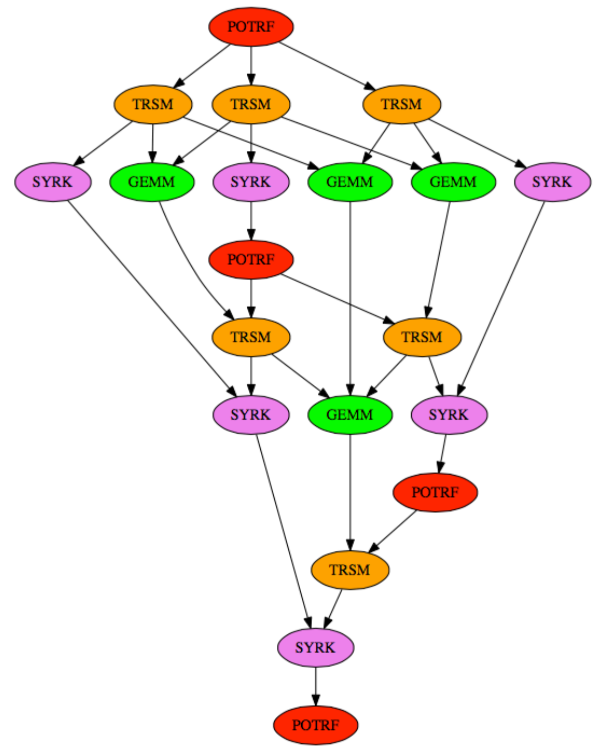
```
#pragma omp task [clause[[,] clause] ...] new-line
    structured-block
```

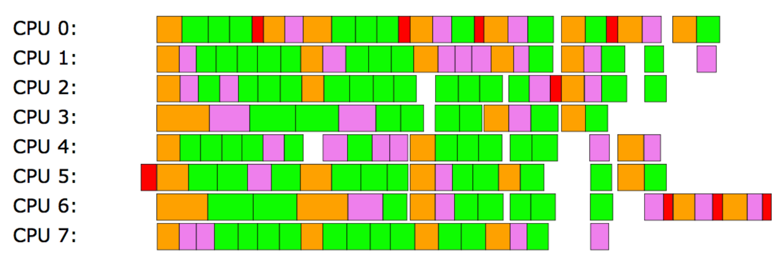
where *clause* is one of the following:

```
if (scalar-expression)
final (scalar-expression)
untied
default(shared | none)
mergeable
private (list)
firstprivate (list)
shared (list)
depend (dependence-type : list)
```

Cholesky Pseudocode

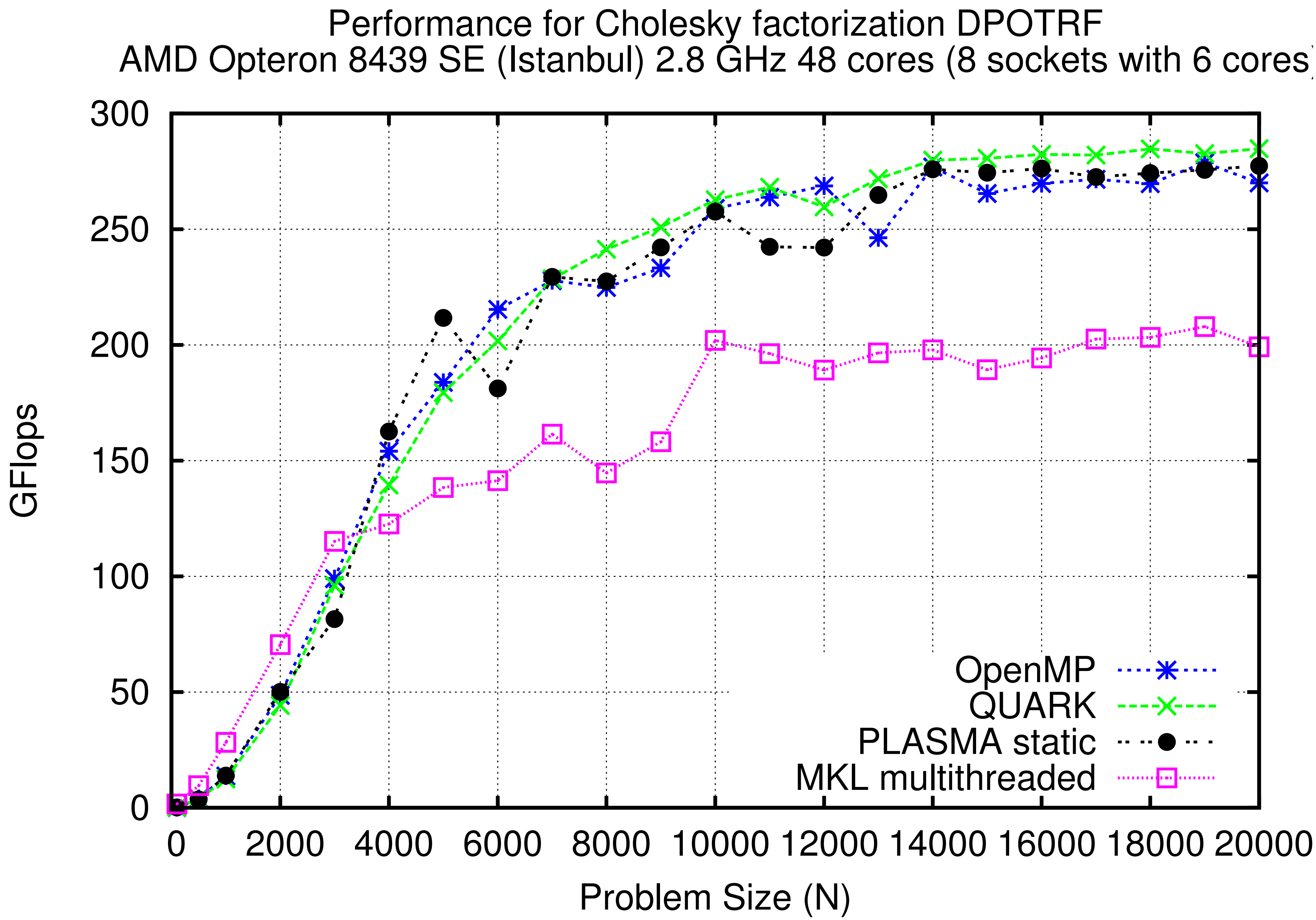
```
#pragma omp parallel
#pragma omp master
{  CHOLESKY( A );  }
CHOLESKY( A ) {
    for (k = 0; k < M; k++) {
        #pragma omp task depend(inout:A(k,k)[0:tilesize]
        {  POTRF( A(k,k) );  }
        for (m = k+1; m < M; m++) {
            #pragma omp task \
                depend(in:A(k,k)[0:tilesize]) \
                depend(inout:A(m,k)[0:tilesize])
            { TRSM( A(k,k), A(m,k) ); }
        }
        for (m = k+1; m < M; m++) {
            #pragma omp task \
                depend(in:A(m,k)[0:tilesize]) \
                depend(inout:A(m,m)[0:tilesize])
            { SYRK( A(m,k), A(m,m) ); }
            for (n = k+1; n < m; n++) {
                #pragma omp task \
                    depend(in:A(m,k)[0:tilesize], \
                              A(n,k)[0:tilesize]) \
                    depend(inout:A(m,n)[0:tilesize])
                {  GEMM( A(m,k), A(n,k), A(m,n) ); }
            }
        }
    }
}
```

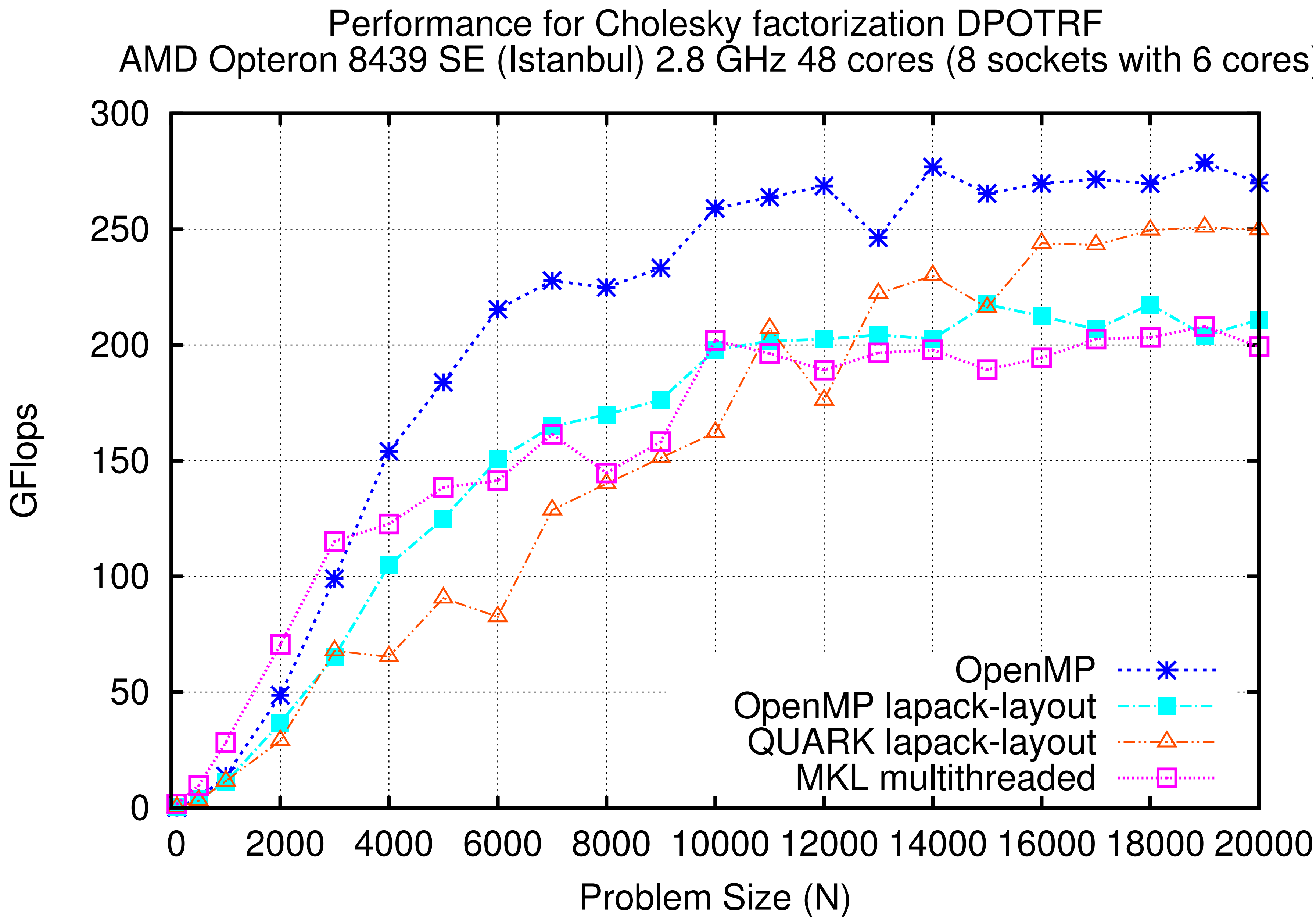




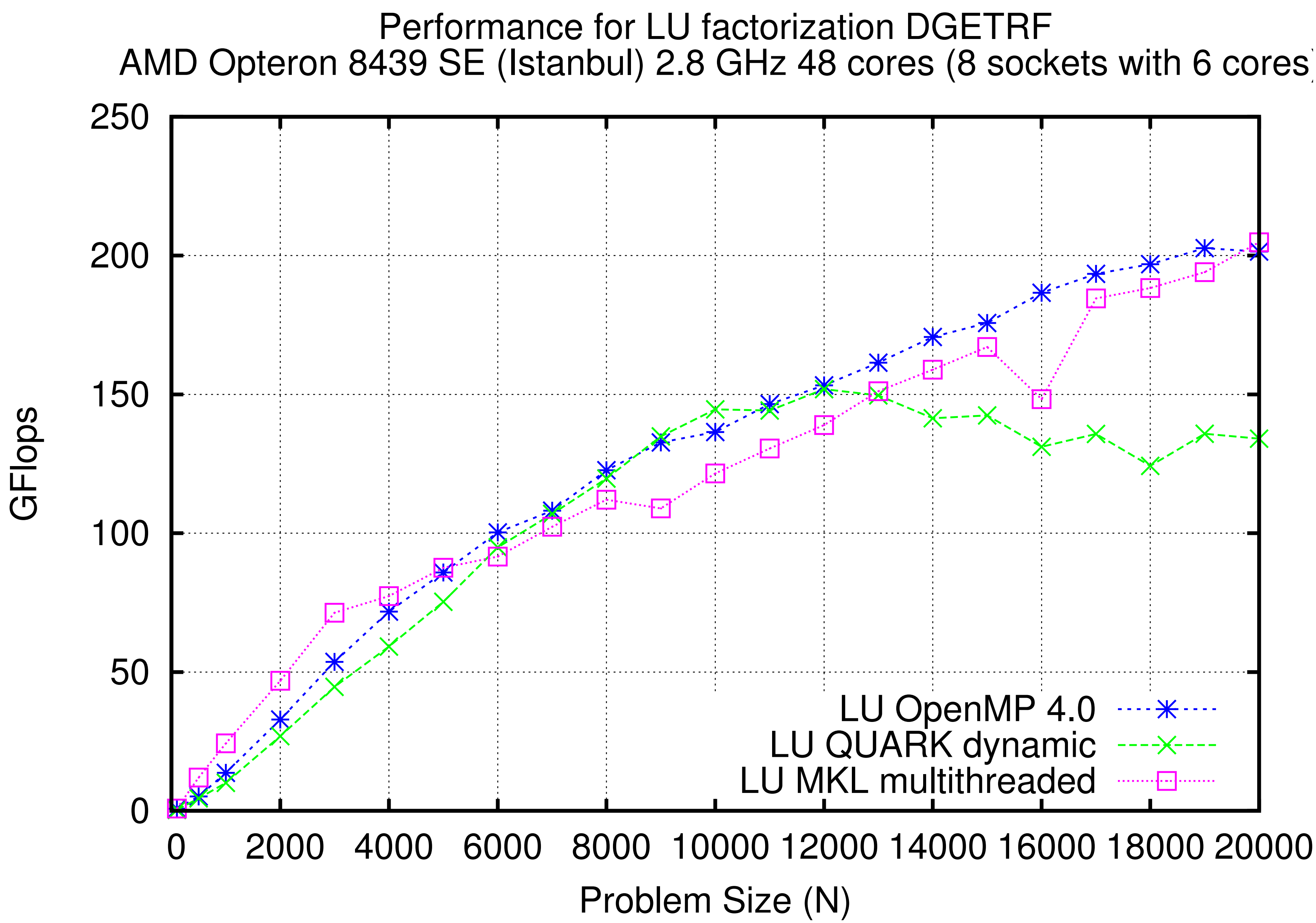
Performance of OpenMP Tasks

- gcc 4.9+ implements OpenMP 4.0
  - Released mid-2014
- icc supports OpenMP 4.0… but slow
- Testing on ig
  - 8 socket, 6 cores per socket, 48 cores
  - Want to check on multi-socket NUMA machine
- OpenMP implementations
  - Cholesky, recursive LU, QR, Cholesky inversion, multi-task dgemm
- Performance
  - Cholesky on tiled layout
  - Cholesky on lapack layout (out-of-place layout translation)
  - LU on tiled layout
  - Cholesky with lapack layout on haswell [dancer]

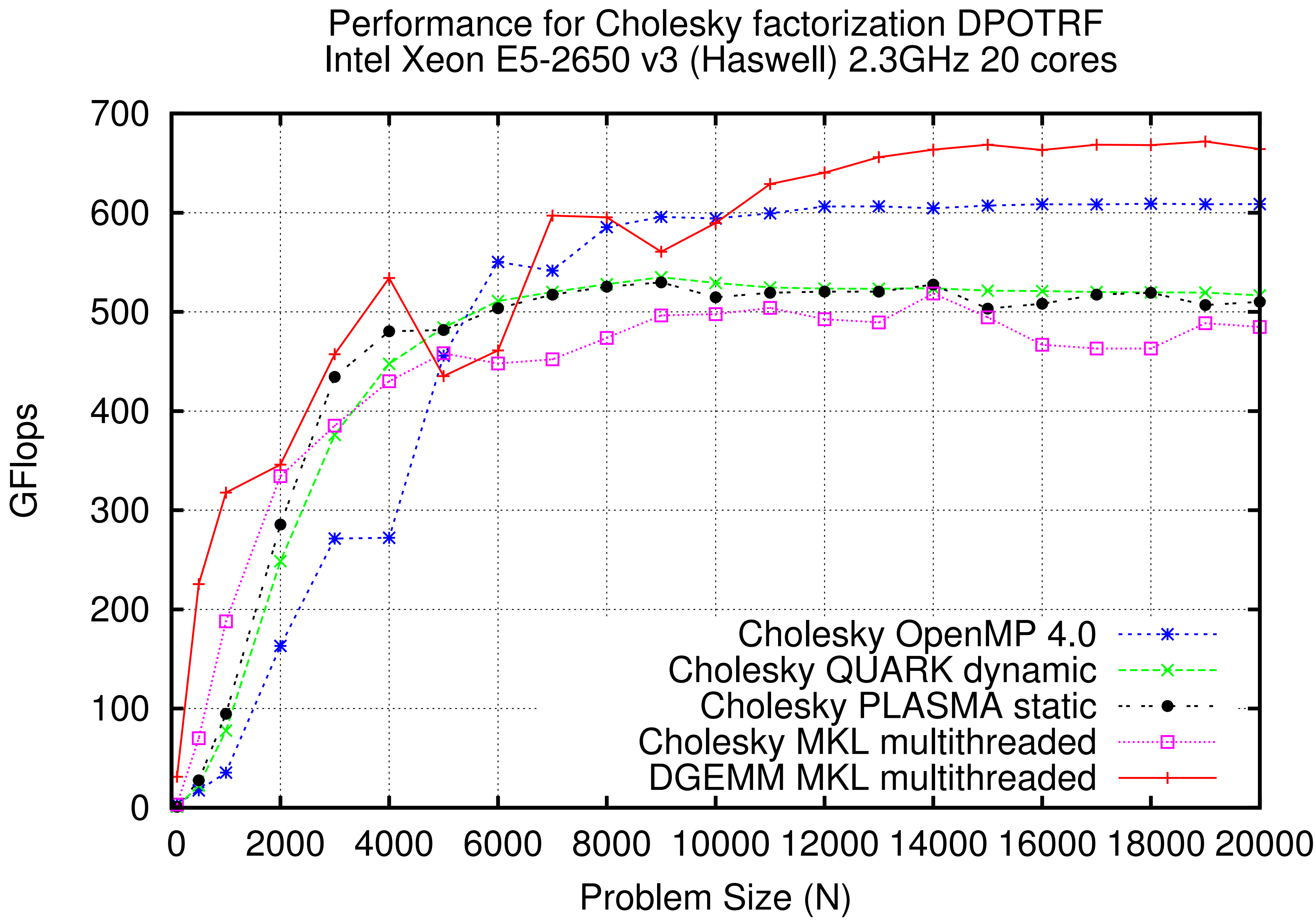

Performance for Cholesky factorization DPOTRF
AMD Opteron 8439 SE (Istanbul) 2.8 GHz 48 cores (8 sockets with 6 cores)
GFlops
300
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Problem Size (N)
OpenMP
QUARK
PLASMA static
MKL multithreaded



Performance for Cholesky factorization DPOTRF
AMD Opteron 8439 SE (Istanbul) 2.8 GHz 48 cores (8 sockets with 6 cores)
GFlops
0
50
100
150
200
250
300
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Problem Size (N)
OpenMP
OpenMP lapack-layout
QUARK lapack-layout
MKL multithreaded

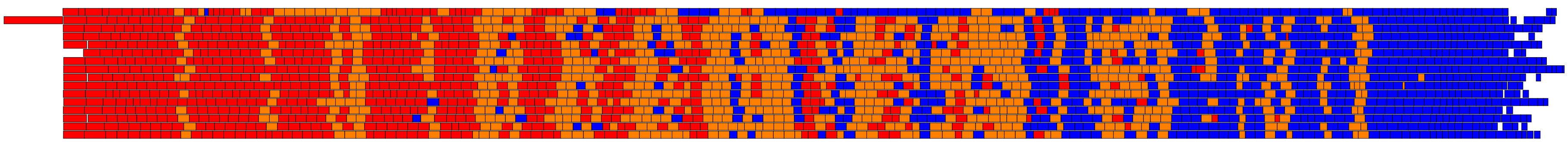


Performance for LU factorization DGETRF
AMD Opteron 8439 SE (Istanbul) 2.8 GHz 48 cores (8 sockets with 6 cores)
GFlops
250
200
150
100
50
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Problem Size (N)
LU OpenMP 4.0
LU QUARK dynamic
LU MKL multithreaded


Performance for Cholesky factorization DPOTRF
Intel Xeon E5-2650 v3 (Haswell) 2.3GHz 20 cores
GFlops
0
100
200
300
400
500
600
700
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
Problem Size (N)
Cholesky OpenMP 4.0
Cholesky QUARK dynamic
Cholesky PLASMA static
Cholesky MKL multithreaded
DGEMM MKL multithreaded

Cholesky Inversion

- 3 routines (potrf, trtri, lauum)
- Not fork-join – merged traces



Lessons and Lumps

- Multithreaded tasks
  - Handled in Recursive LU by setting a ReadWrite dummy on the data
  - Insert multiple tasks with Read on data
  - Set ReadWrite dummy on data
- Control task locality for data
  - Cannot be managed
  - It is possible to bind threads to cores (export OMP_PLACES)
  - BUT, tasks cannot be assigned to specific threads
- Binding tasks for accelerators/MPI
  - OpenMP defines “target” to support some accelerators
    - gcc claims support for Xeon Phi and new Nvidia GPUs
    - intel supports MIC/Xeon Phi

Lessons and Lumps

- Priority for tasks (critical path)
  - Put into the to-be-release OpenMP 4.1 standard
- Using barriers to speed up code
  - In Cholesky (lapack layout)

    parallel out-of-place lapack-to-tile layout translation

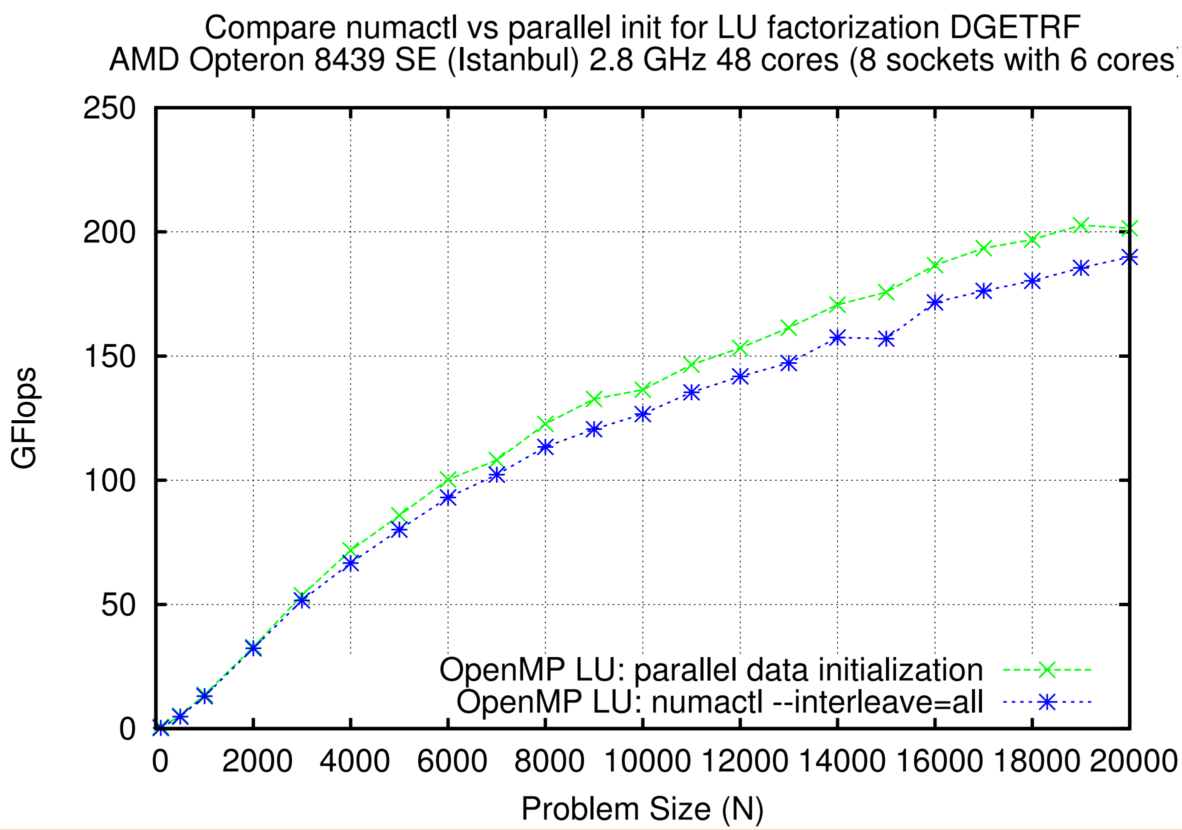
    (add a #pragma omp taskwait here for 5-10% better runs)

    parallel tile cholesky solve

    parallel tile-to-lapack layout translation
- Lock process to threads (ala hwloc)
  - Use export OMP_PROC_BIND [CLOSE, SPREAD, MASTER, TRUE]
  - #pragma omp parallel proc_bind(close)
    - 5-10% improvement for linear algebra

Lessons and Lumps

- Initializing with numactl vs parallel
  - Much better to initialize tiles in parallel (5-10%)
  - Question: Undercounting MKL? (since numactl was used)



Review of OpenMP Requirements

- Performance
  - Good, matching QUARK in most places
- Multi-threaded tasks
  - Works for recursive LU so far
- Complex merged algorithms
  - Cholesky inversion works. Need eigenvector/eigenvalue work
- Data Regions
  - Upper/Lower triangle
  - Use manual data copies for read items. Just need a few for lookahead
- Task priorities
  - Missing but expected
- Cancellation
  - Taskgroups can be cancelled, but that would be an entire calculation. Not tested.
- Accumulator/Gatherv
  - Missing and unlikely to be implemented.