Jack Dongarra, Mathieu Faverge, Hatem Ltaief, Piotr Luszczek

High Performance Matrix Inversion Based on LU Factorization for Multicore Architectures

presented by

Piotr Luszczek

ICL UT

Preliminaries

Problem Statement

$$A \in \boldsymbol{R}^{n \times n}$$

$$PA = LU$$

$$U \rightarrow U^{-1}$$

$$L \rightarrow L^{-1}$$

$$A^{-1} \in \boldsymbol{R}^{n \times n}$$

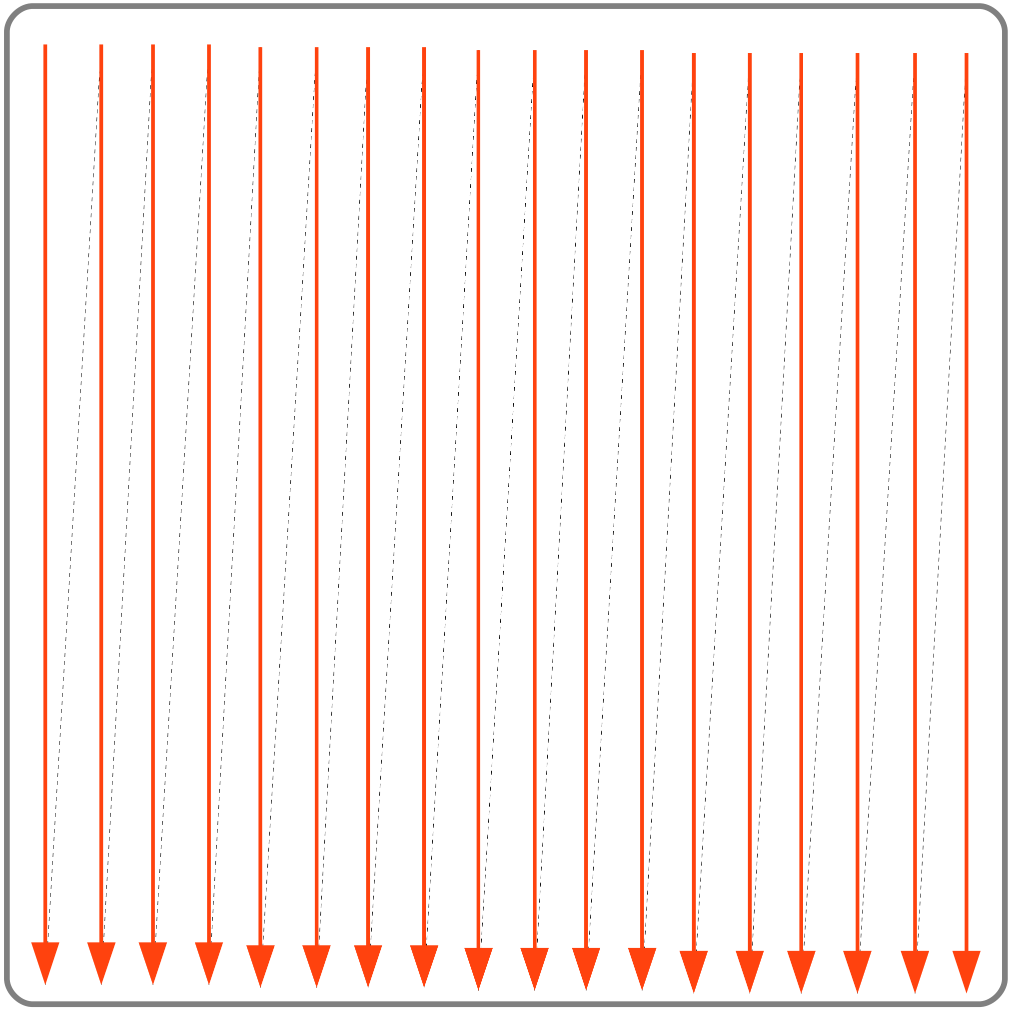
To Keep in Mind...

In the vast majority
of
practical computational problems,
it is unnecessary
and
inadvisable
to
actually compute A^{-1}.

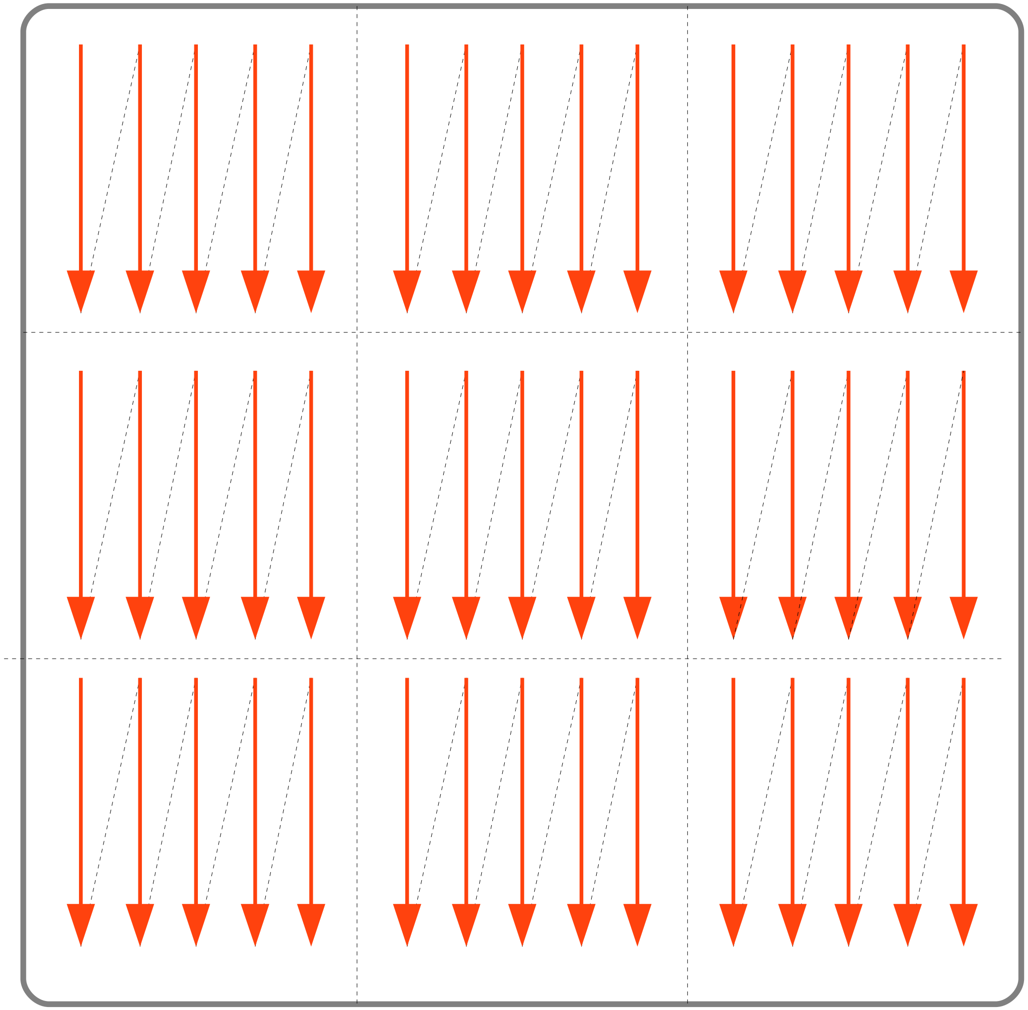
Forsythe, Malcolm, and Moler

Data Layouts for Matrix Elements

Column-major (LAPACK and derivatives)



Tile (PLASMA)



Tasks and DAGs

Block LU Inversion

- For each panel *LU factorization*
  DGETF2( )
  DLASWP( )
  DLASWP( )
  DTRSM( )
  DGEMM( )
- For each panel *Invert U*
  DTRMM( )
  DTRSM( )
  DTRTI2( )
- For each panel *Invert L*
  DLACPY( )
  DLASET( )
  DGEMM( )
  DTRSM( )
- DLASWP( ) *column interchanges*

Tile LU Inversion

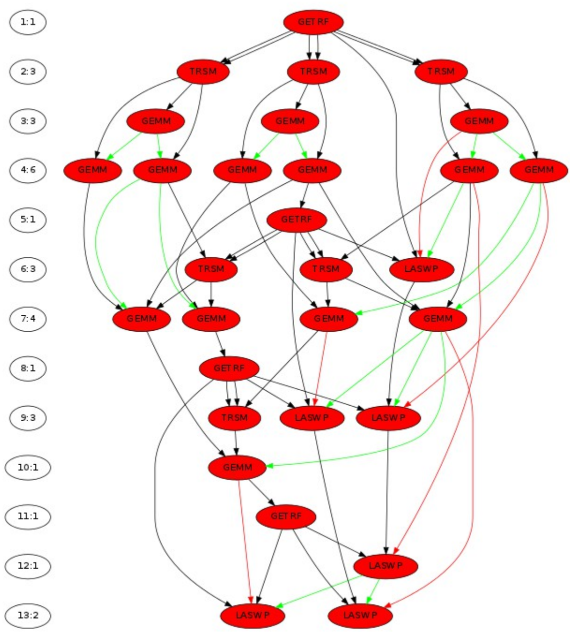
- For each diagonal tile
  -DGETRFR() *parallel recursive LU*
  for each tail tile panel
  -DLASWP( )
  for each tail tile
  -DGEMM( )
  for each left tile panel
  -DLASWP( )
- For each diagonal tile *Invert U*
  for each tile in panel
  -DTRSM( )
  for each tail tile
  -DGEMM( )
  for each left panel tile
  -DTRSM( )
  -DTRTRI( )
- For each left tile *Invert L*
  -DLACPY( )
  -DLASET( )
  ...

ICL UT

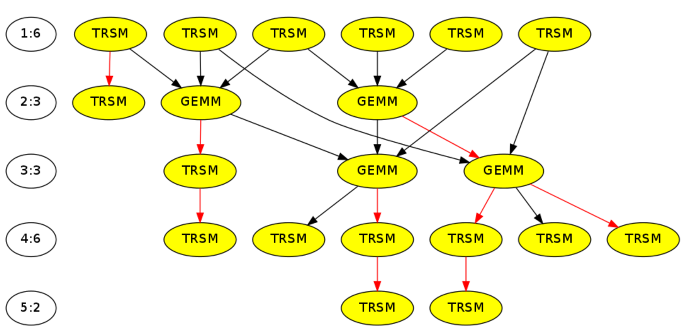
Queuing Functions with QUARK

```
QUARK_Insert_Task(
panel_LU_task,
M, matrix_1 , INPUT,
N, matrix_2 , INOUT,
1, result , OUTPUT,
K, buffer , SCRATCH,
0);
```

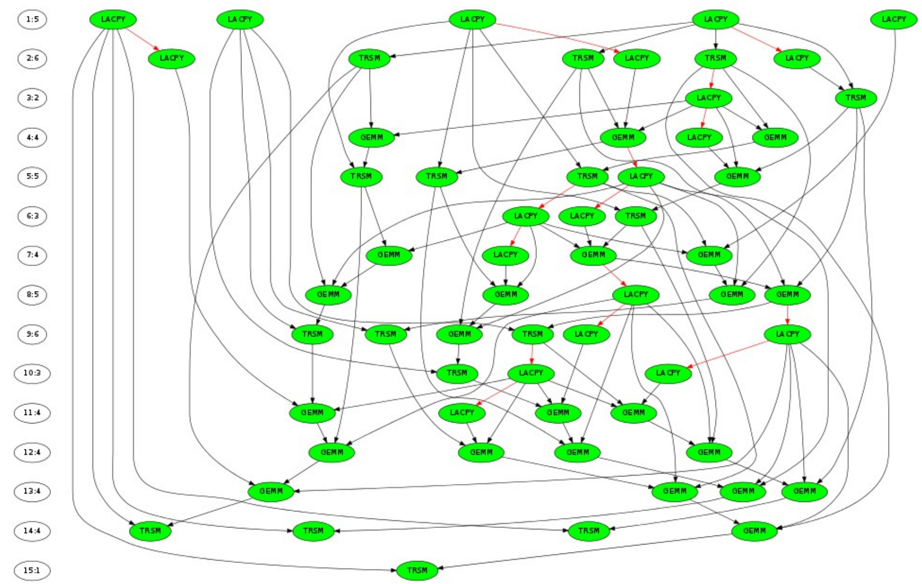
DAGs of Tasks, Each State Separately



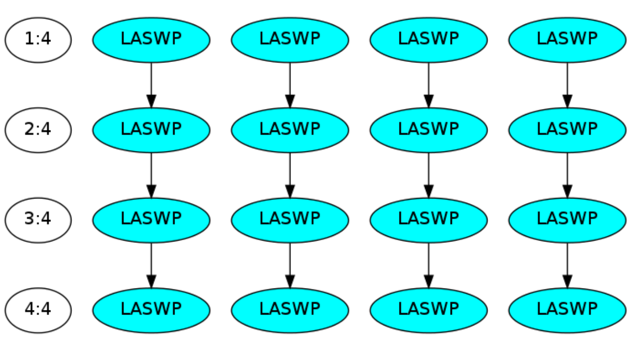
1 – LU Factorization



2 – Computation of L^{-1}

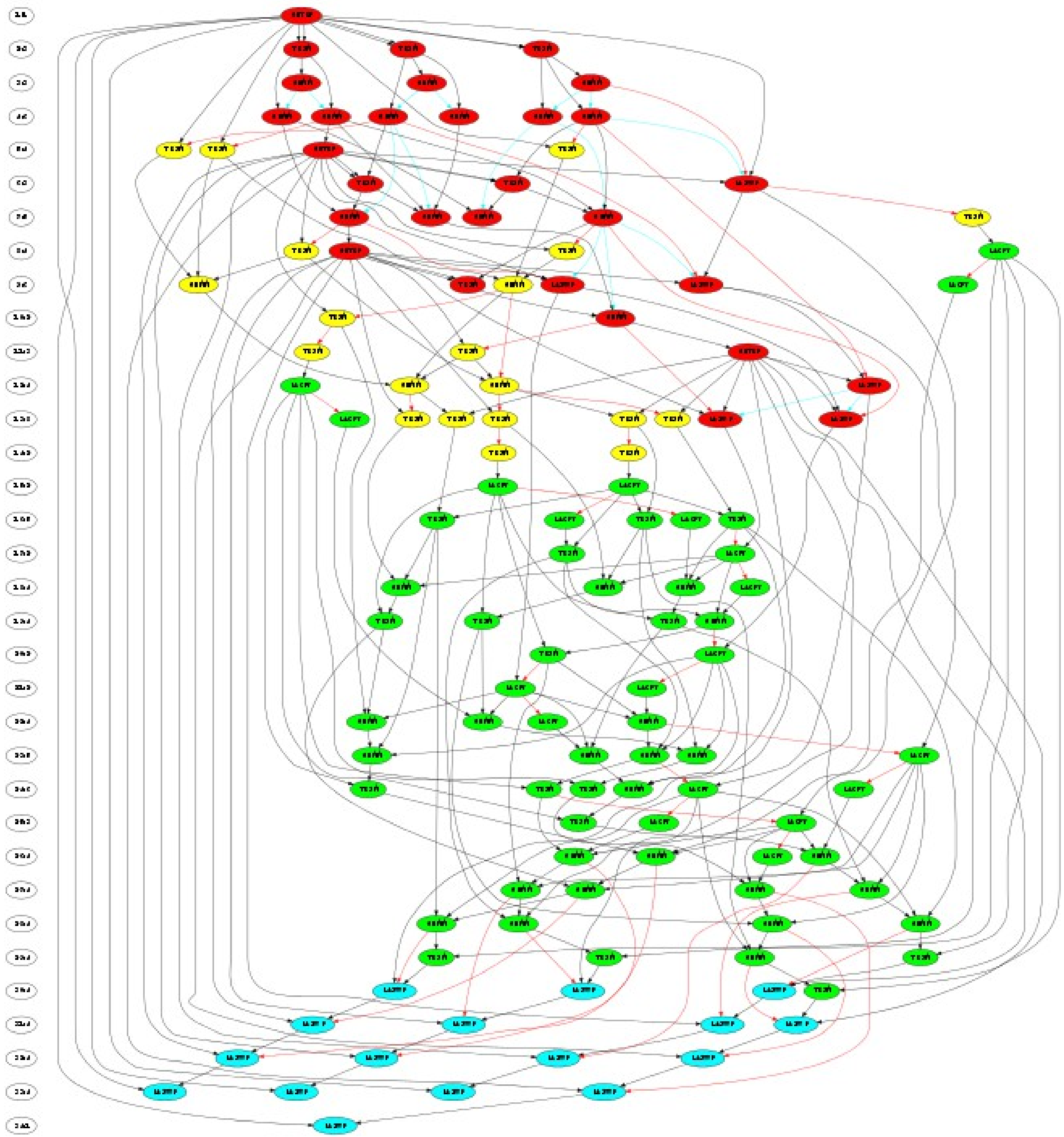


3 – Computation of U^{-1}



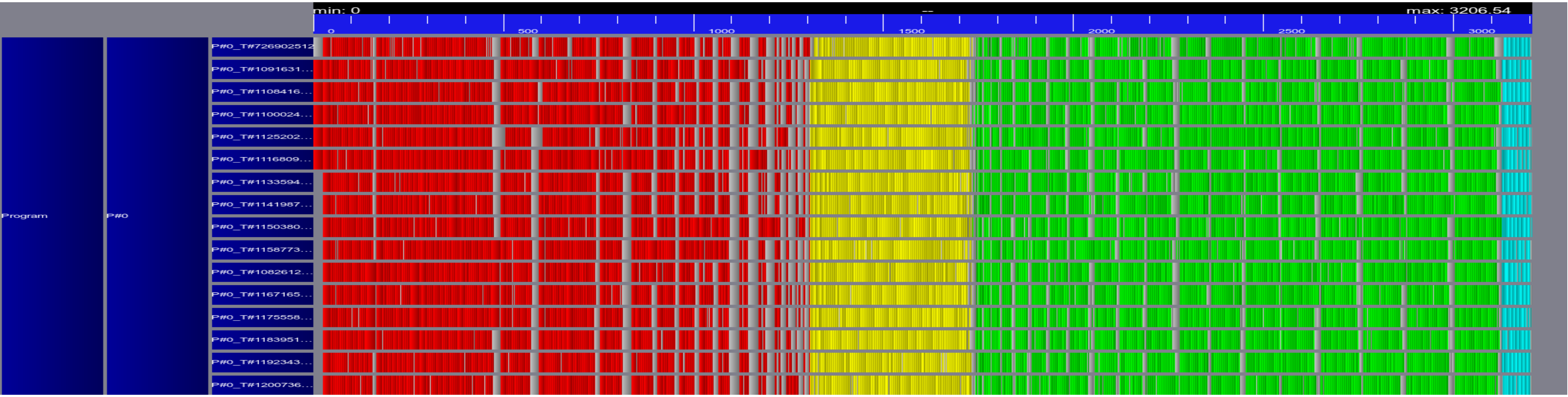
4 – Column swapping

DAGs of Tasks, All Stages Overlapped

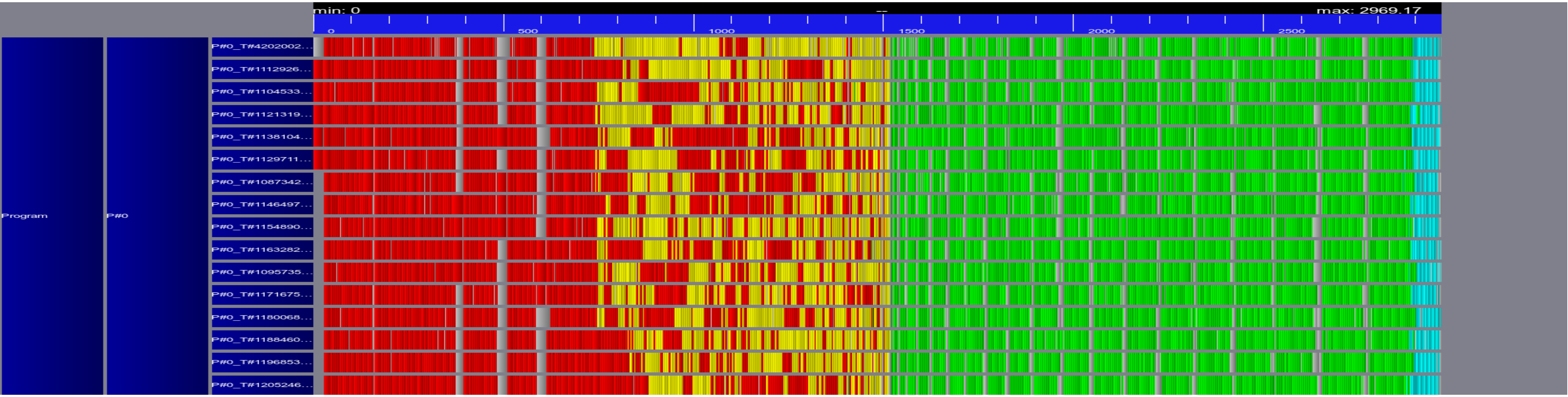


Execution Traces

No Overlap of Stages

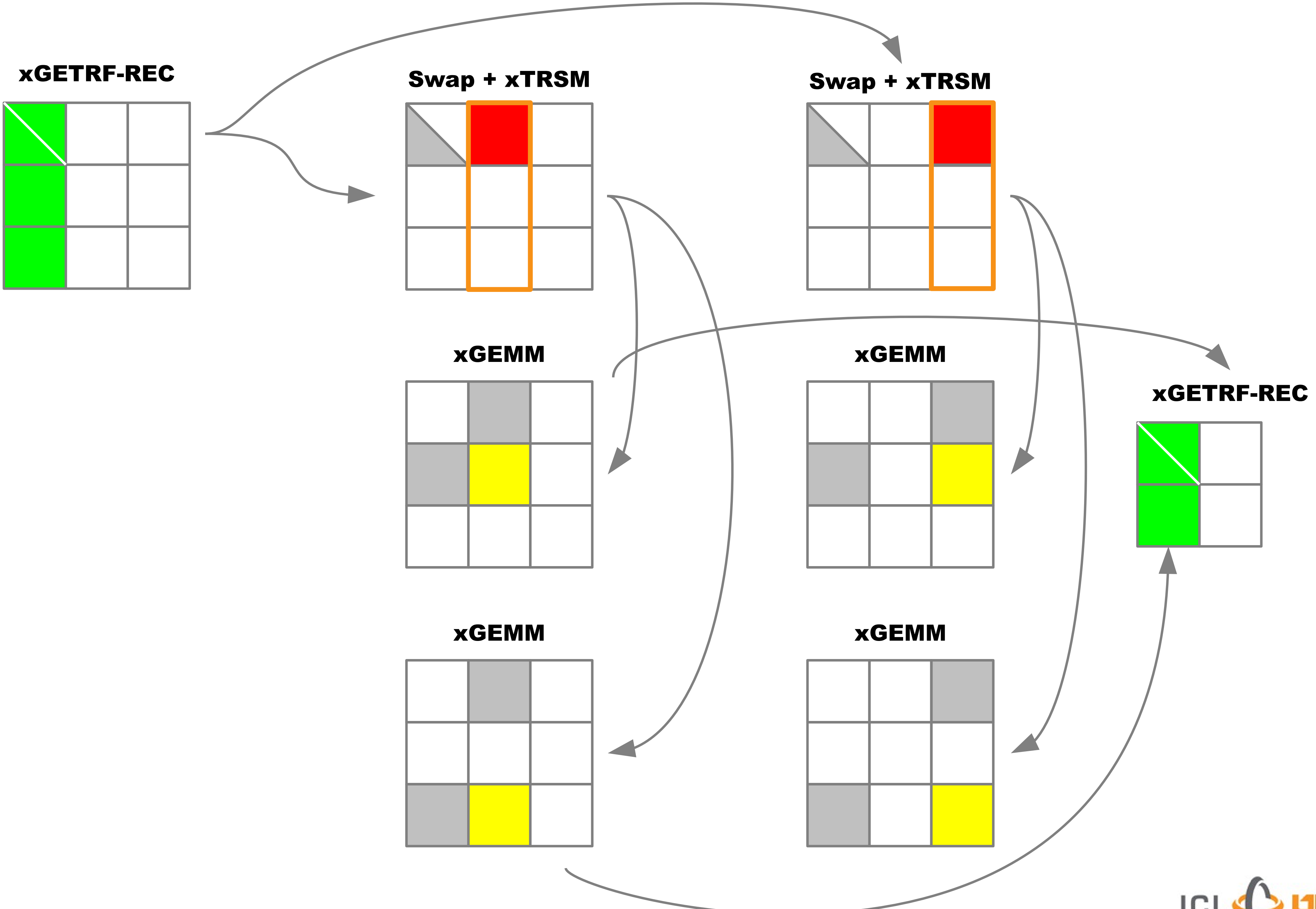


Overlap of Stages

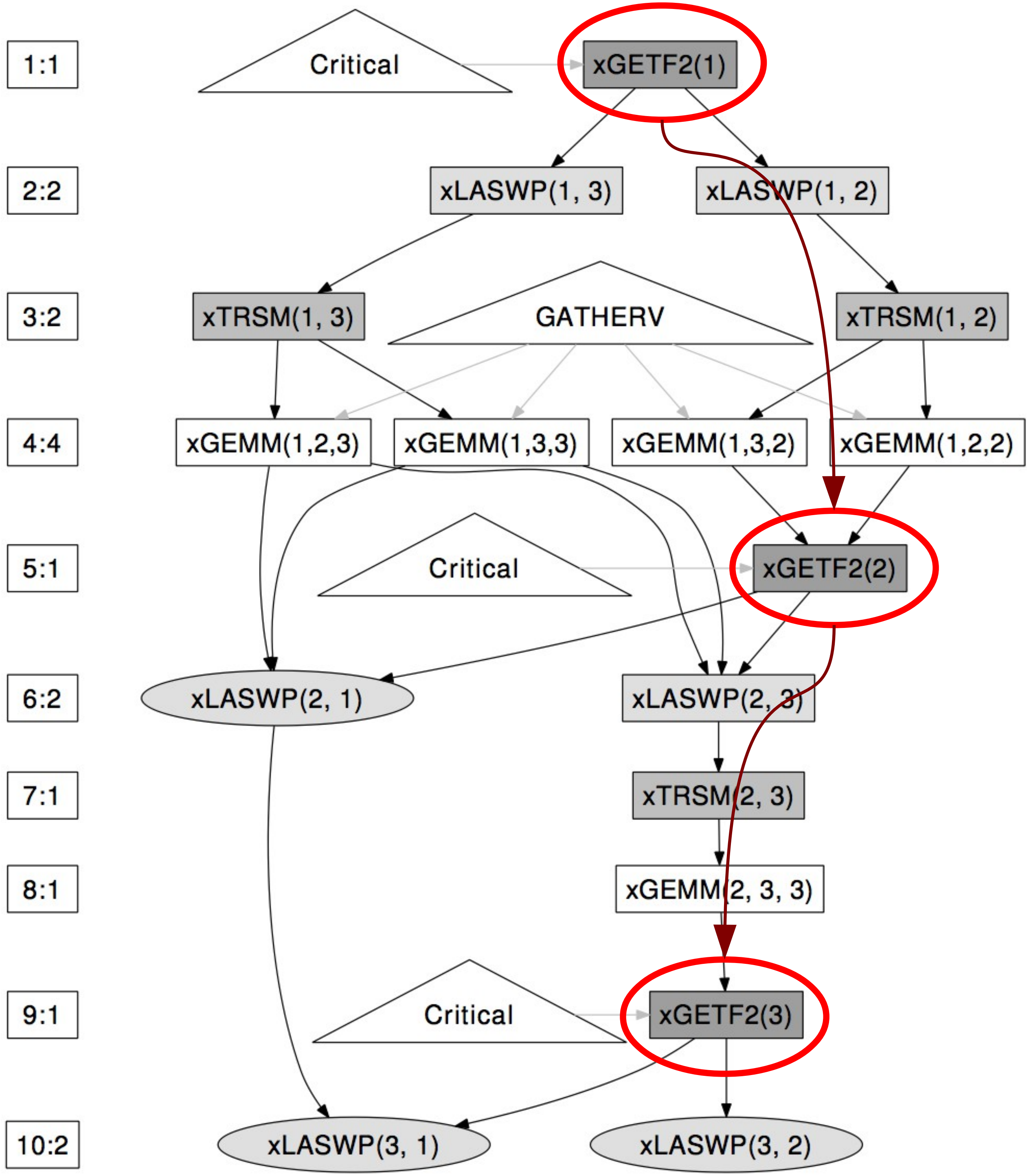


The Case for Nested Parallelism

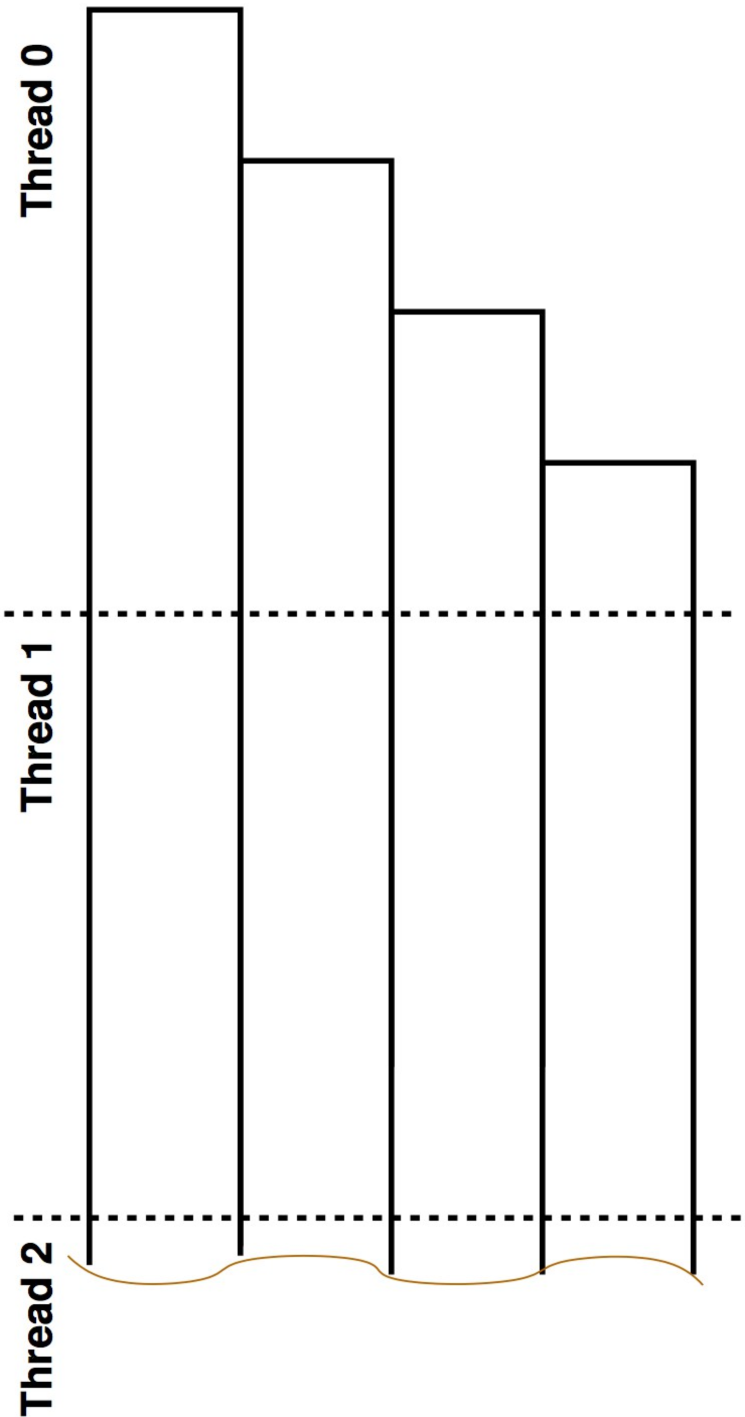
Panel Factorization as the Sequential Bottleneck



Panel Factorization is On Critical Path of DAG



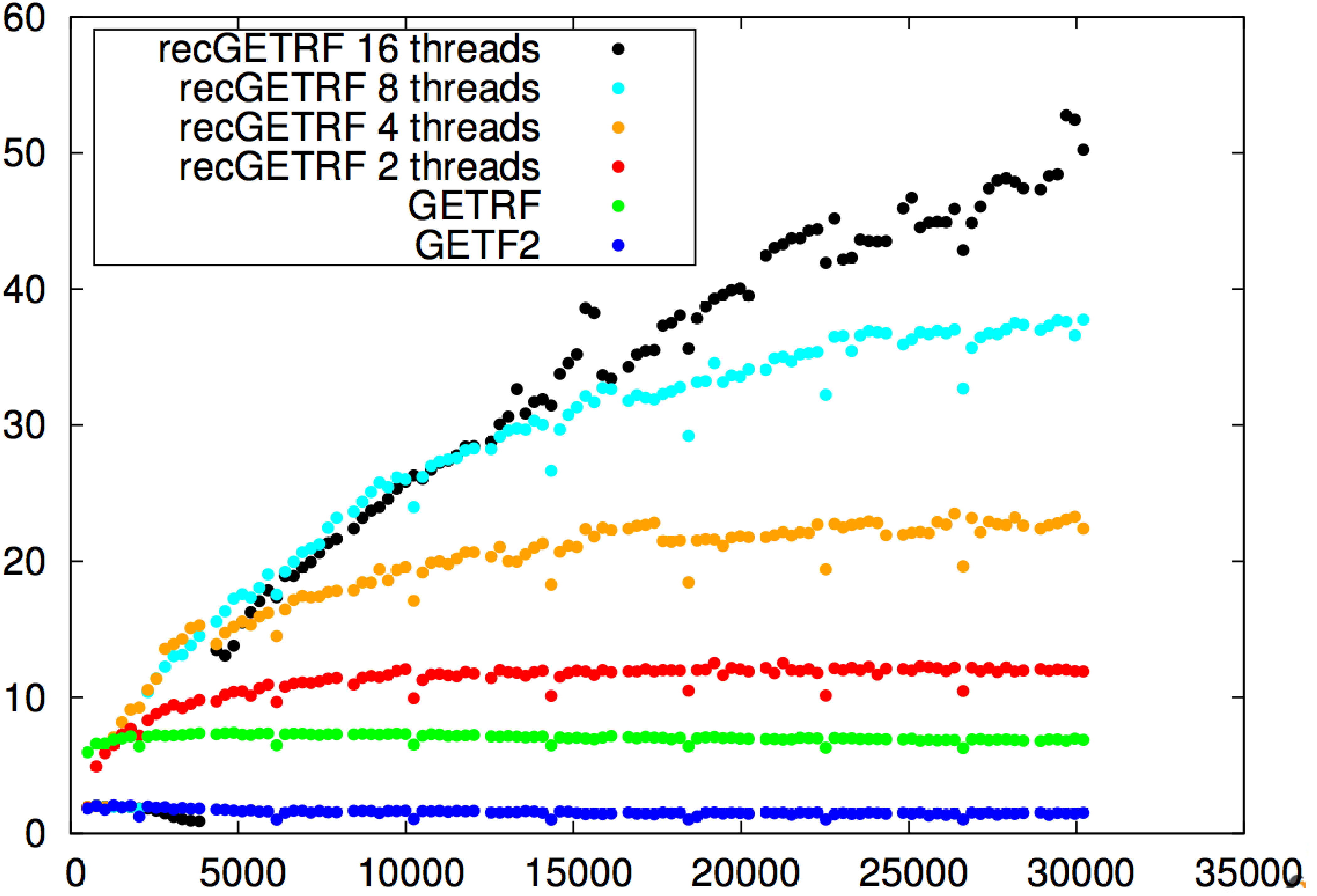
Parallel Panel Factorization: Data Partitioning



Parallel Panel Factorization: Algorithm

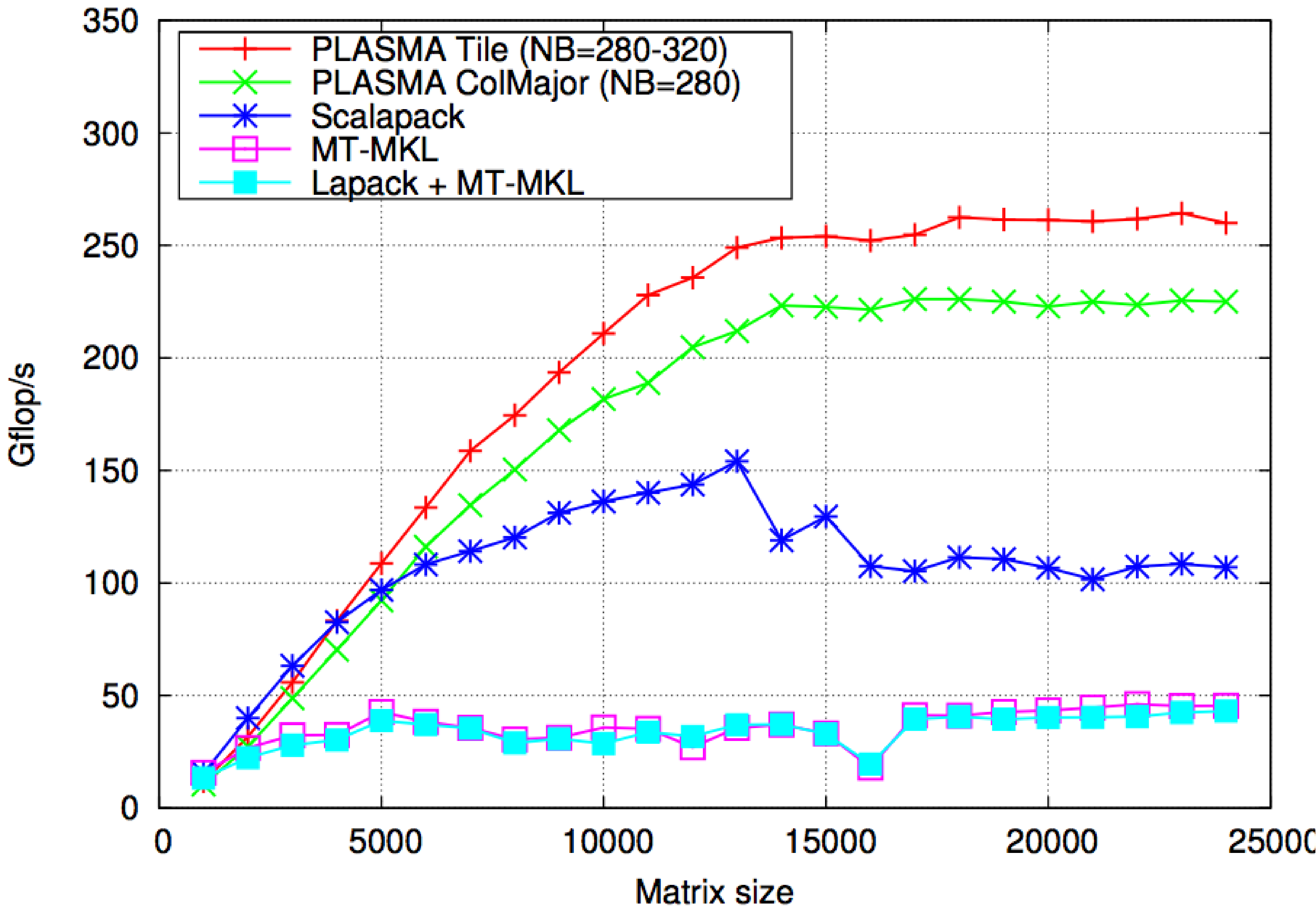
```
function xGETRFR(M, N, column) {
    if N == 1 {                              single column, recursion stops
        idx = split_IxAMAX(...)              compute local maximum of modulus
        gidx = combine_IxAMAX(idx)           combine local results
        split_xSCAL(...)                     scale local data
    } else {
        xGETRFR(M, N/2, column)              recursive call to factor left half
        xLASWP(...)                          pivoting forward
        split_xTRSM(...)                     triangular solve
        split_xGEMM(...)                     Schur's complement
        xGETRFR(M, N-N/2, column+N/2)        recursive call to factor right half
        xLASWP(...)                          pivoting backward
    }
}
```

Quick Performance Experiment

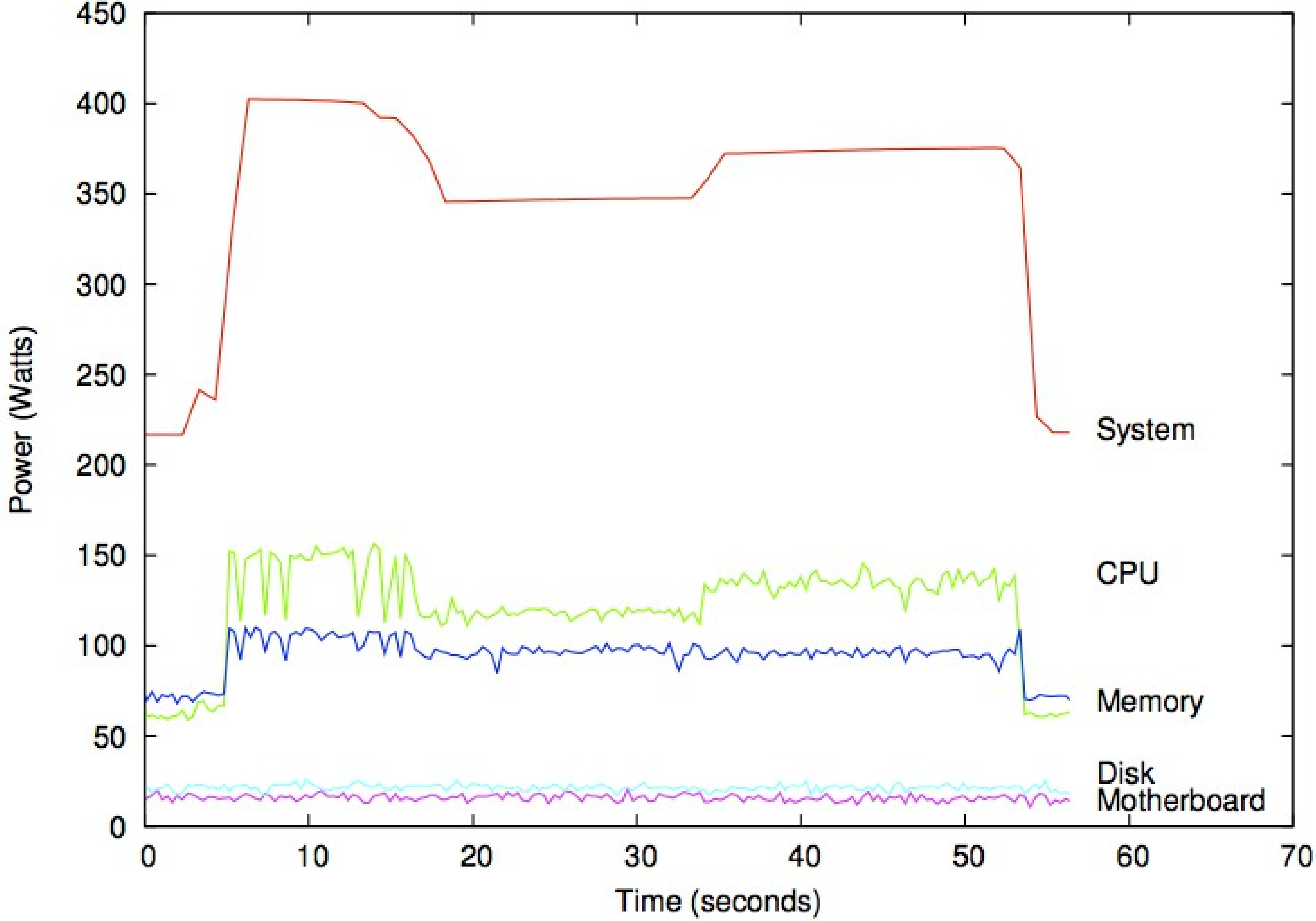


Results

Performance on AMD MagnyCours, 4x12=48 cores

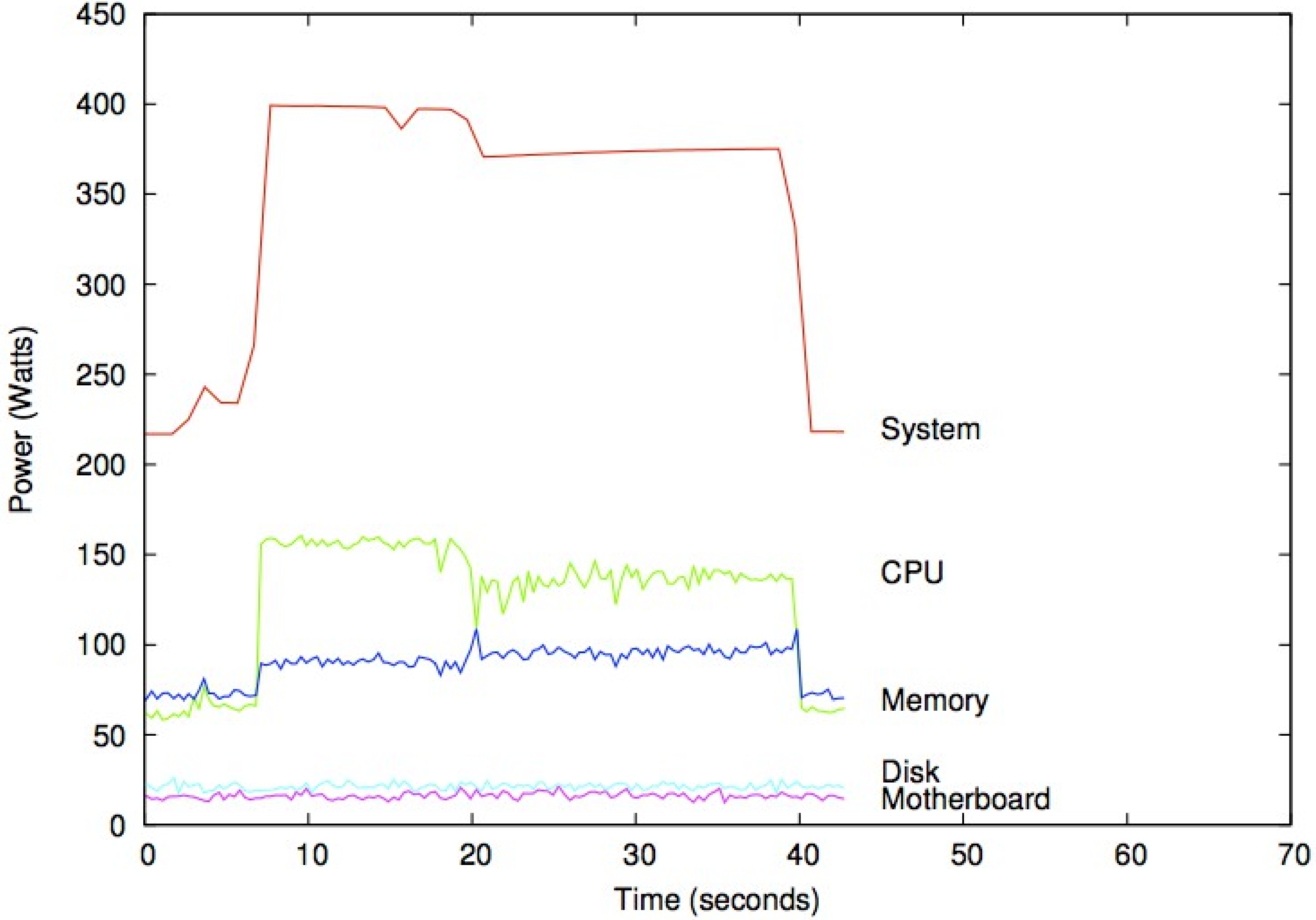


LU Inversion's Power Profile: LAPACK

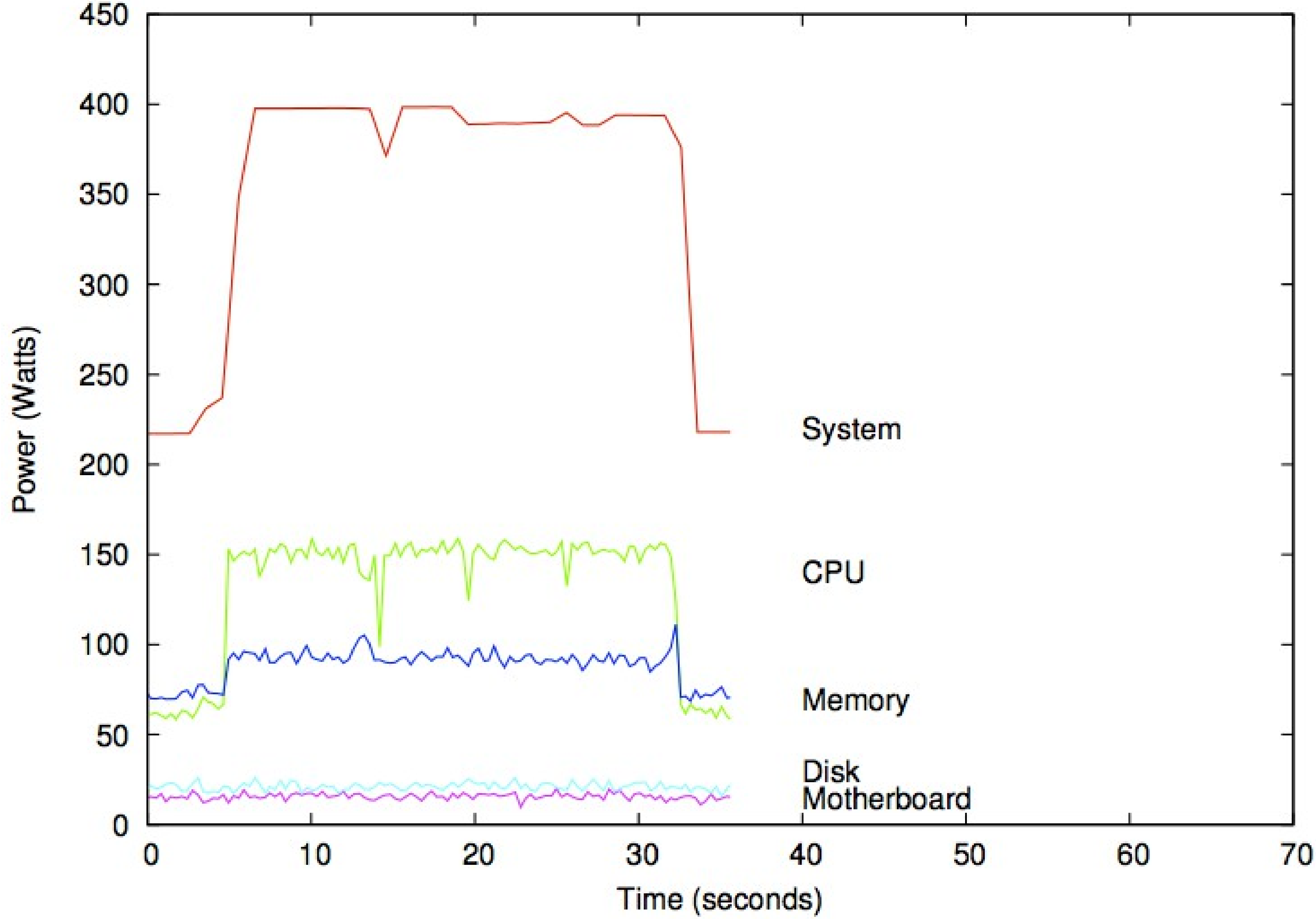


ICL UT

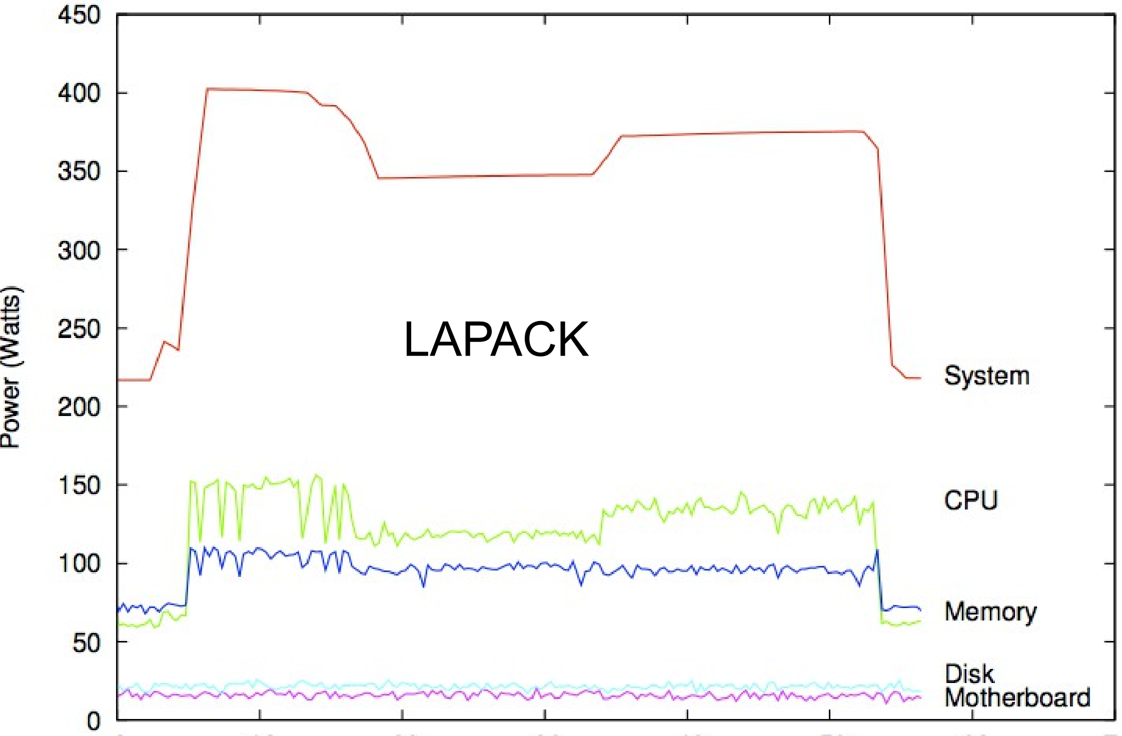
LU Inversion's Power Profile: MKL

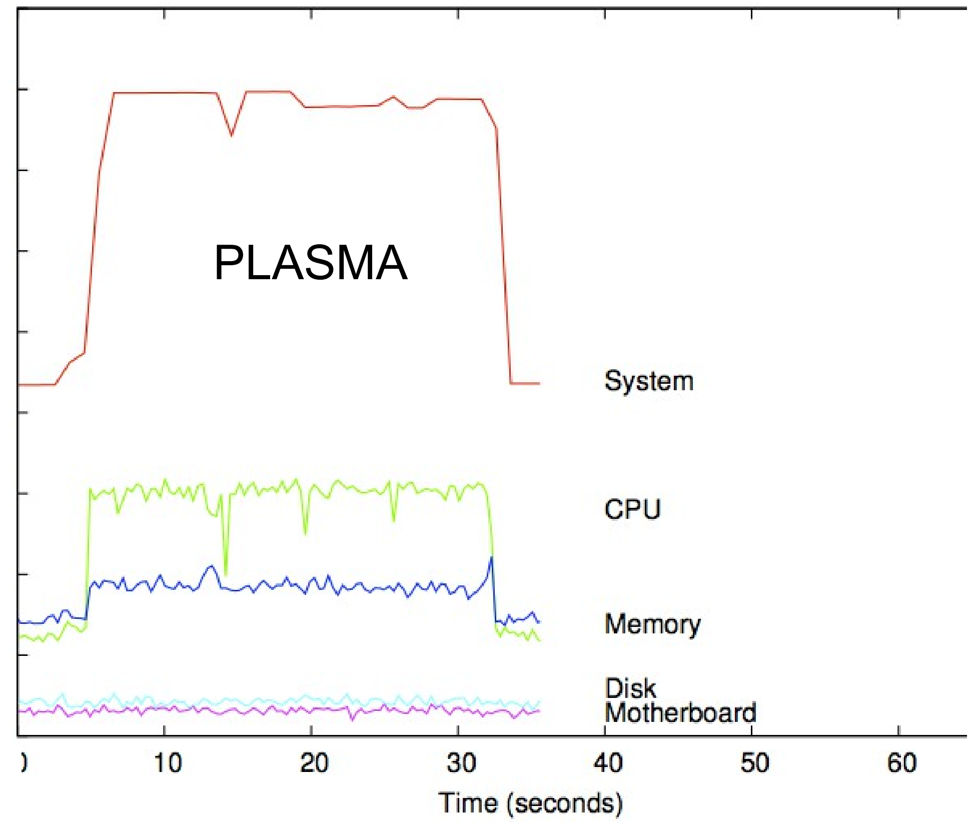


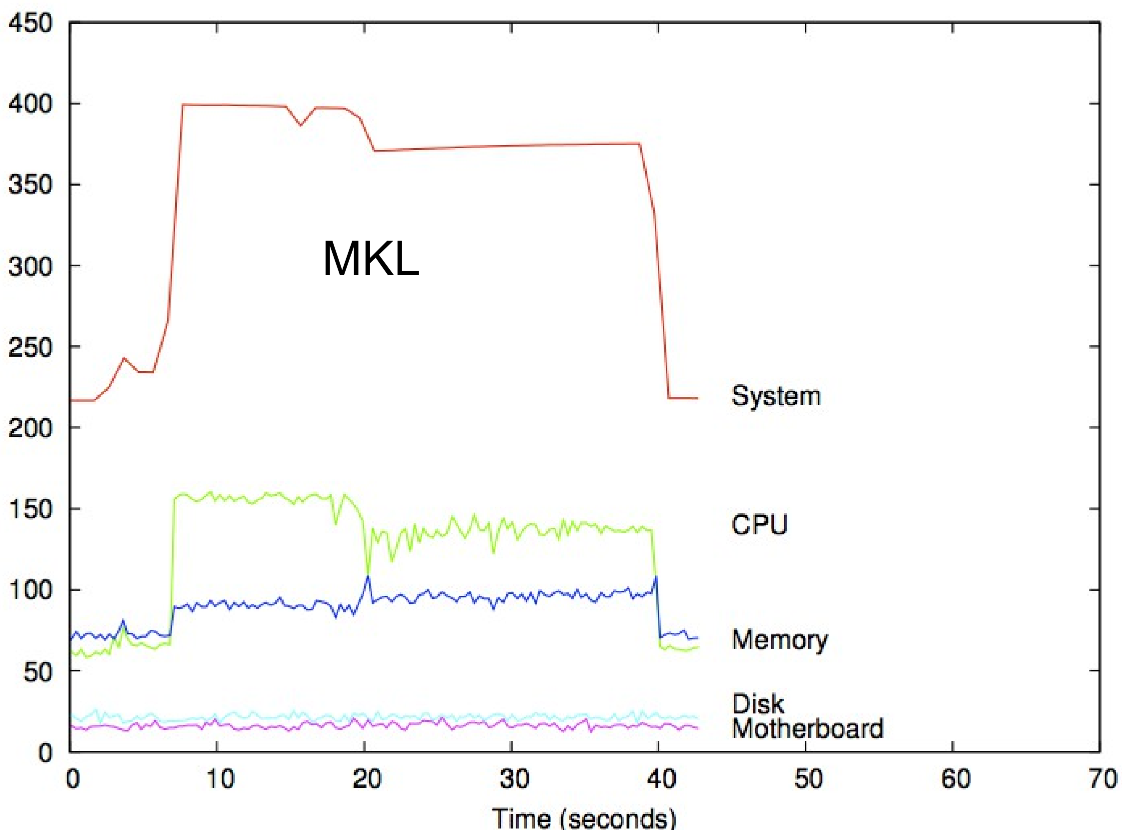
LU Inversion's Power Profile: PLASMA



ICL UT







This work was sponsored by NSF, DOE, and Microsoft