Dynamic Task Discovery in PaRSEC

Reazul Hoque

Overview

- PaRSEC
- Dynamic Task Discovery
- Features
- Validations
- Results
- Future Work

PaRSEC and Current Approach

- PaRSEC – Parallel Runtime Scheduling and Execution Controller.
- Runtime capable of scheduling tasks.
- Tasks can be generated by multiple higher level domain specific interface. e.g. PTG
- Parameterized task graph (PTG) is a compressed representation of DAG.
- Pre-compiler compiles the PTG into C code and then using symbolic evaluation the execution and the dependency tracking between tasks are maintained.

Sample JDF

GEMM(m, n, k)

```
READ  A <- C TRSM(m, k)
READ  B <- C TRSM(n, k)
RW    C <- (k == 0)   ? dataA(m, n)  : C GEMM(m, n, k-1)
          -> (n == k+1) ? C TRSM(m, n) : C GEMM(m, n, k+1)
```

Dynamic Task Discovery

- New interface for PaRSEC.
- Expressing dependencies is easier.
- Builds the DAG dynamically during runtime.
- The DAG is unrolled in memory, whereas in the other approach it is not.
- Parameter packing and the format of the insertion in similar to other runtimes like QUARK, StarPu etc.

QUARK

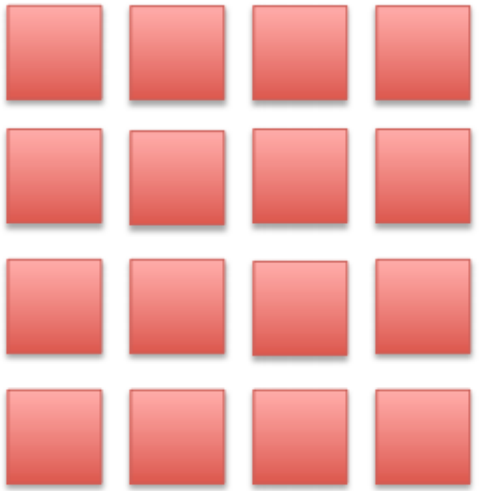
```
for (k = 0; k < A.mt; k++) {
    QUARK_Insert_Task(quark, CORE_zpotrf_quark, task_flags,
                sizeof(int),                        &n,         VALUE,
                sizeof(PLASMA_Complex64_t)*nb*nb,    A(k,k),                INOUT,
                0);
        for (m = k+1; m < A.mt; m++) {
            QUARK_Insert_Task(quark, CORE_ztrsm_quark, task_flags,
                              sizeof(int),                        &m,         VALUE,
                              sizeof(int),                        &n,         VALUE,
                              sizeof(PLASMA_Complex64_t)*nb*nb,    A(k,k),                INPUT,
                              sizeof(PLASMA_Comp  lex64_t)*nb*nb,  A(m,k),                INOUT,
                              0);

        }
        for (m = k+1; m < A.mt; m++) {
            QUARK_Insert_Task(quark, CORE_zherk_quark, task_flags,
                              sizeof(int),                        &n,         VALUE,
                              sizeof(int),                        &k,         VALUE,
                              sizeof(PLASMA_Complex64_t)*nb*nb,    A(m,k),                INPUT,
                              sizeof(PLASMA_Complex64_t)*nb*nb,    A(m,m),                INOUT,
                              0);

            for (n = k+1; n < m; n++) {
                QUARK_Insert_Task(quark, CORE_zgemm_quark, task_flags,
                                  sizeof(int),                        &m,         VALUE,
                                  sizeof(int),                        &n,         VALUE,
                                  sizeof(int),                        &k,         VALUE,
                                  sizeof(PLASMA_Complex64_t)*nb*nb,    A(m,k),                INPUT,
                                  sizeof(PLASMA_Complex64_t)*nb*nb,    A(n,k),                INPUT,
                                  sizeof(PLASMA_Complex64_t)*nb*nb,    A(m,n),                INOUT,
                                  0);
            }
        }
}
```

PaRSEC

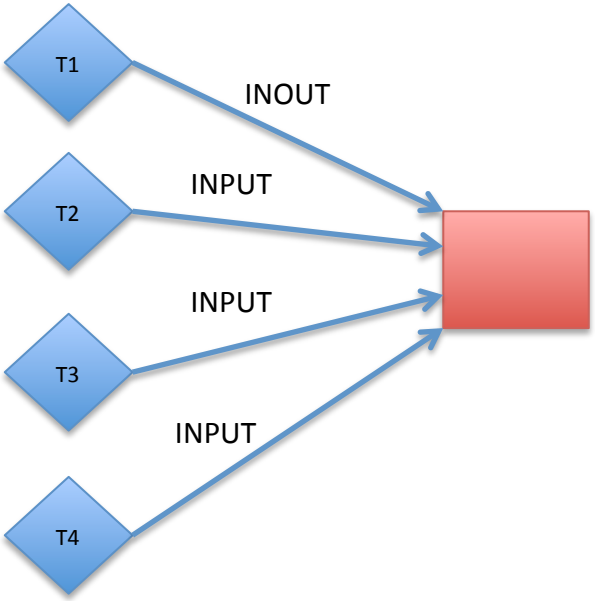
```
for(k=0;k<total/*A.mt*/;k++){
    insert_task_generic_fptr(DAGUE_dtd_handle, call_to_kernel_PO, "Zpotrf",
            sizeof(int),         &tempkm,    VALUE,
            PASSED_BY_REF,       TILE_OF(DAGUE_dtd_handle, A, k, k), INOUT, DEFAULT,
            0);
    for(m=k+1;m<total/*A.mt*/;m++){
        insert_task_generic_fptr(DAGUE_dtd_handle, &call_to_kernel_TR, "Ztrsm",
                sizeof(int),        &tempmm,    VALUE,
                PASSED_BY_REF,      TILE_OF(DAGUE_dtd_handle, A, k, k), INPUT, DEFAULT,
                PASSED_BY_REF,      TILE_OF(DAGUE_dtd_handle, A, m, k), INOUT, DEFAULT,
                0);
    }
    for(m=k+1;m<total/*A.mt*/;m++){
        insert_task_generic_fptr(DAGUE_dtd_handle, &call_to_kernel_HE, "Zherk",
                sizeof(int),      &tempmm,     VALUE,
                PASSED_BY_REF,    TILE_OF(DAGUE_dtd_handle, A, m, k), INPUT, DEFAULT,
                PASSED_BY_REF,    TILE_OF(DAGUE_dtd_handle, A, m, m), INOUT, DEFAULT,
               0);
        for(n=k+1;n<m;n++){
              insert_task_generic_fptr(DAGUE_dtd_handle,  &call_to_kernel_GE, "Zgemm",
                  sizeof(int),      &tempmm,    VALUE,
                  PASSED_BY_REF,    TILE_OF(DAGUE_dtd_handle, A, m, k), INPUT, DEFAULT,
                  PASSED_BY_REF,    TILE_OF(DAGUE_dtd_handle, A, n, k), INPUT, DEFAULT,
                  PASSED_BY_REF,    TILE_OF(DAGUE_dtd_handle, A, m, n), INOUT, DEFAULT,
                  0);
        }
    }
}
```

Data Setup and Dependency

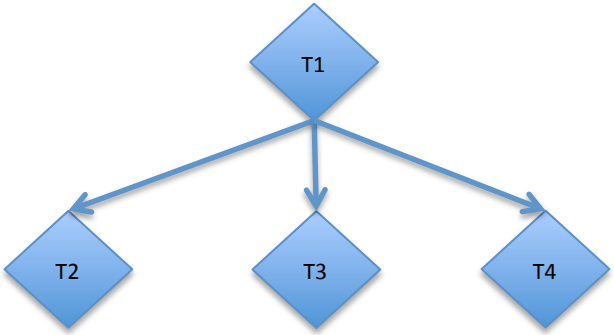
- Matrix is represented by blocks
- Flags to express operation type
  - INPUT
  - OUTPUT
  - INOUT



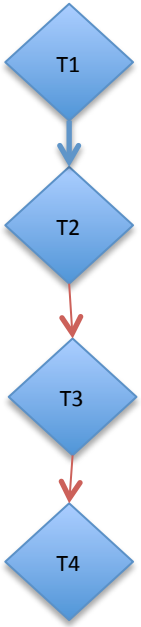
Discovered in given order



Before

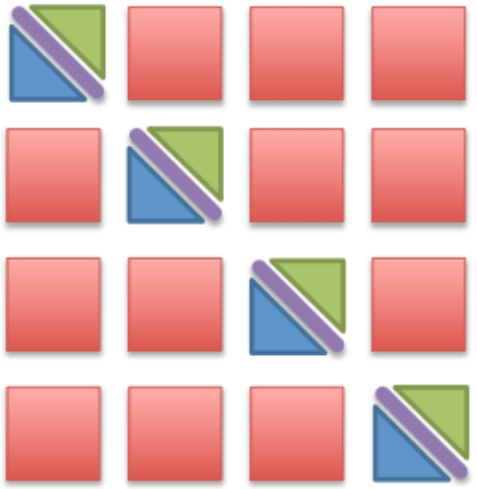


Now



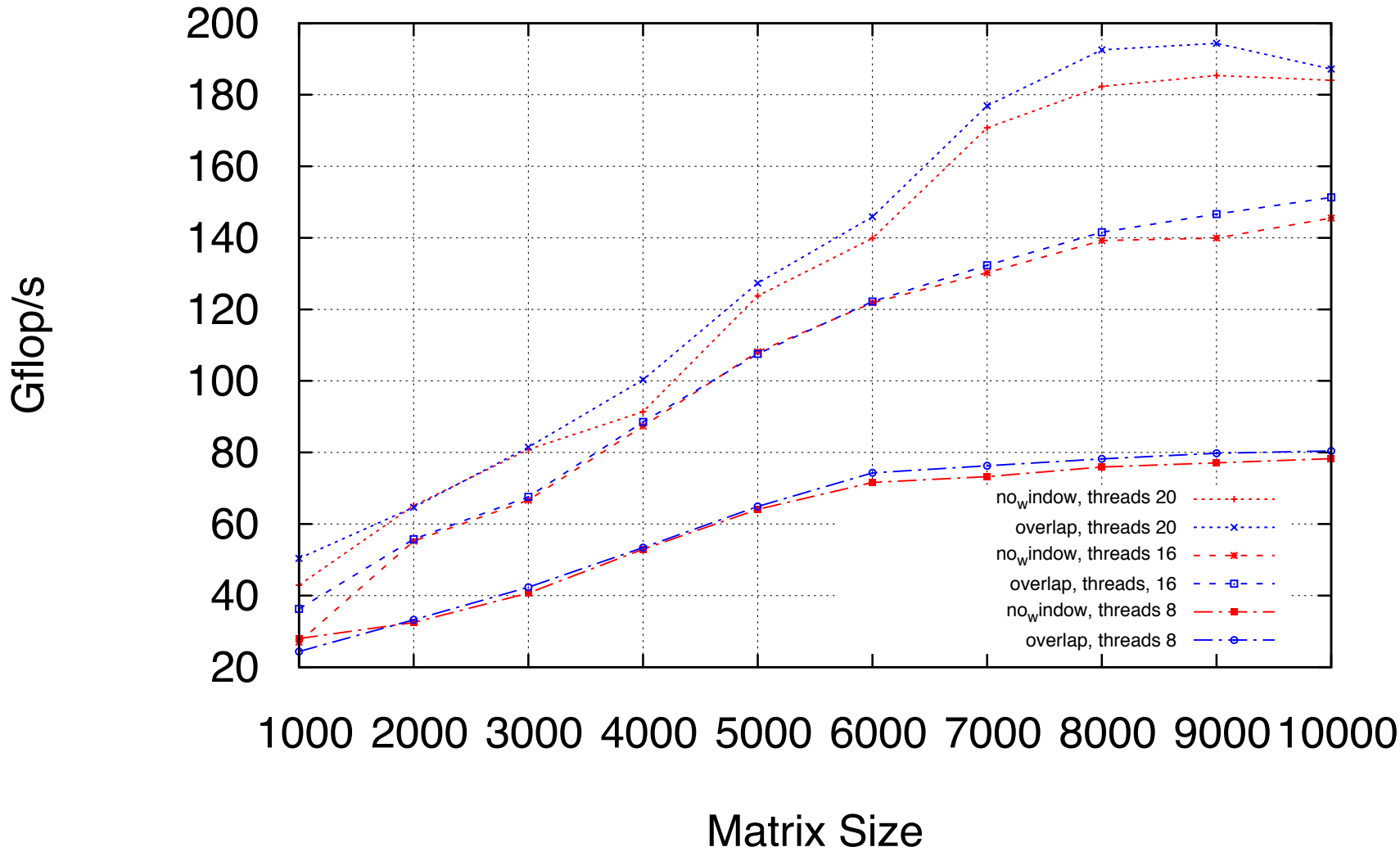
Region Information

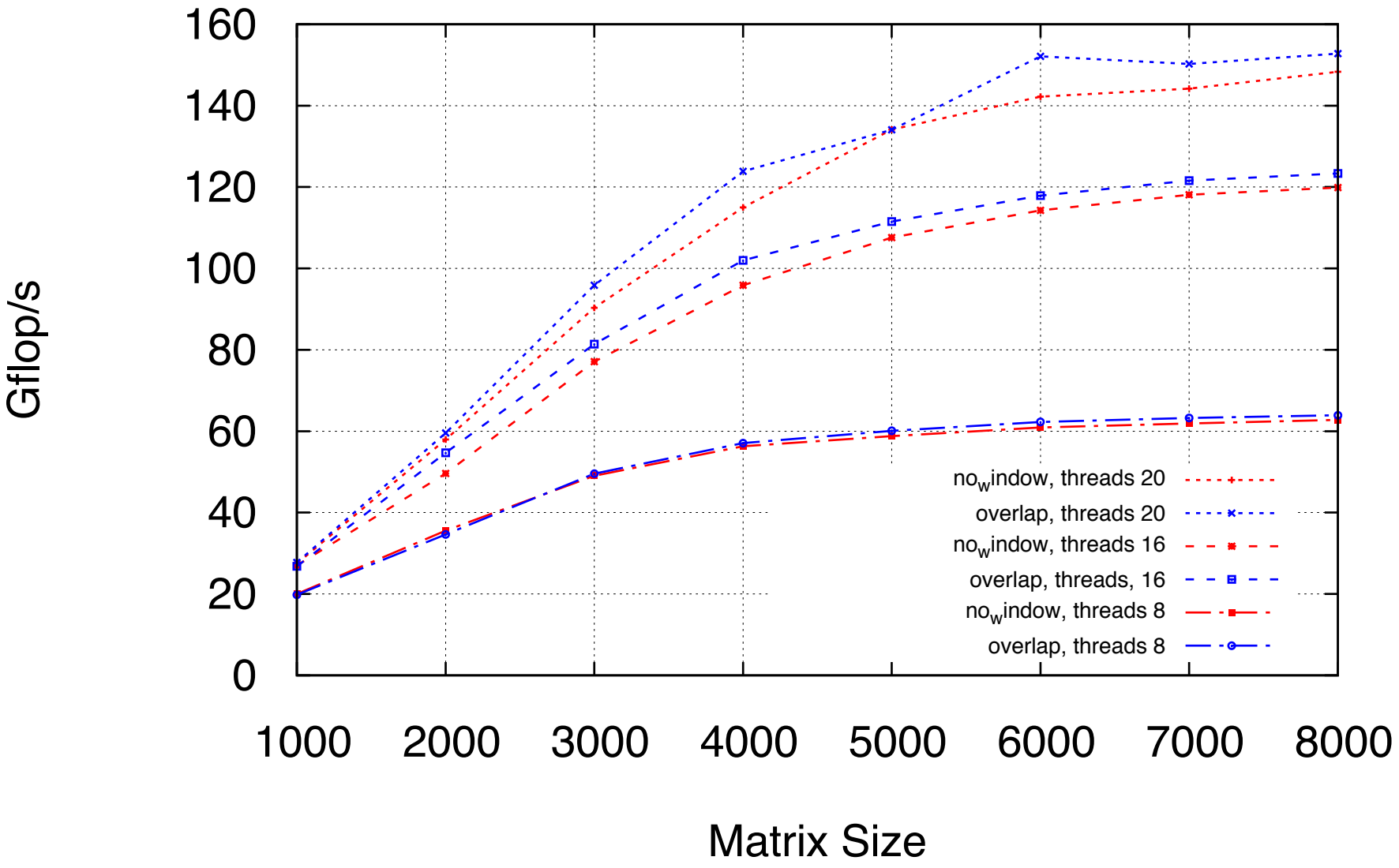
- Some operation touches only part of the block
- Regions:
  - Upper triangle
  - Diagonal
  - Lower Triangle
- Tracking dependence based on the region information increases parallelism.



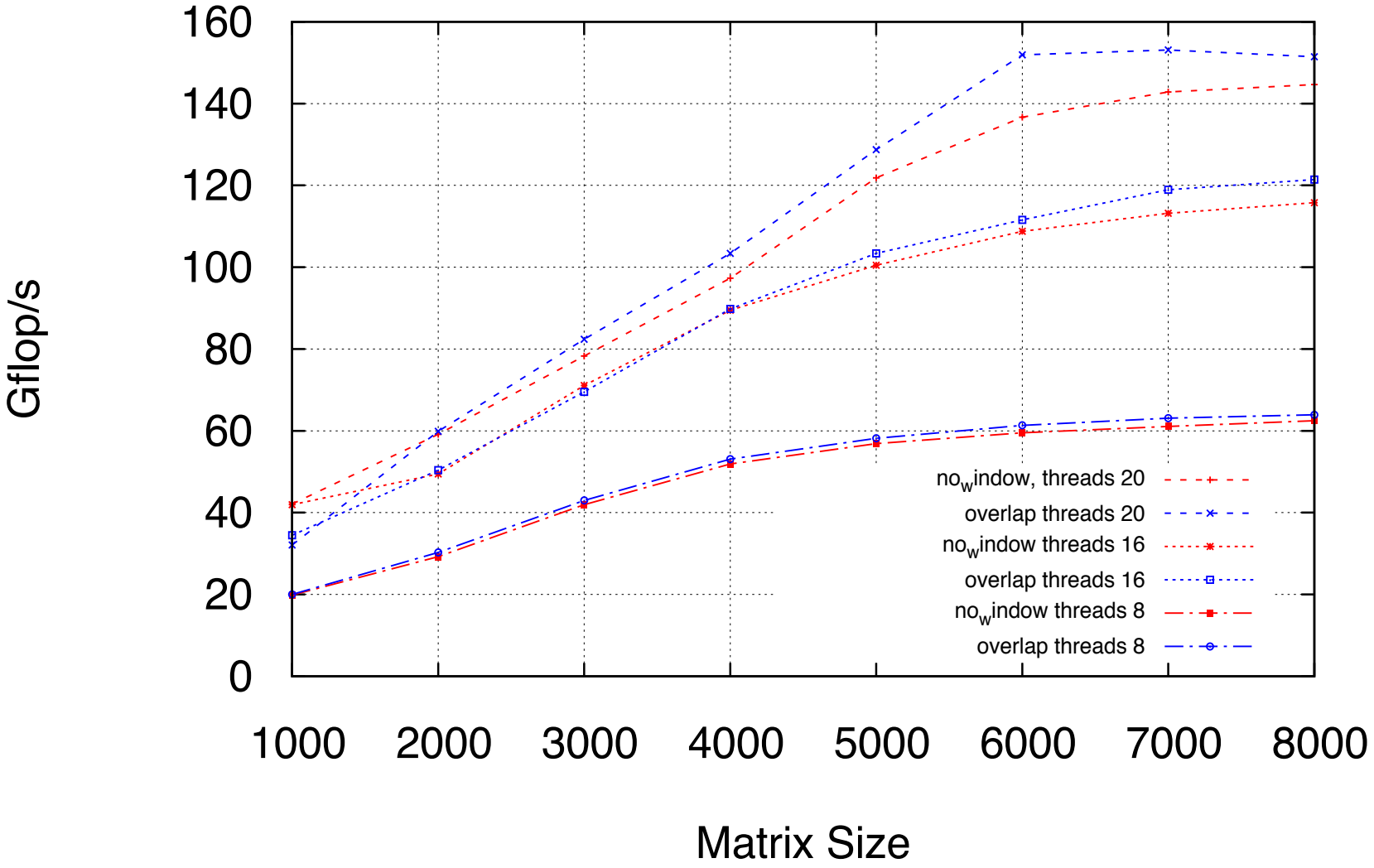
DTD Features

- Overlapping the building of DAG and execution.
- The whole DAG is not unrolled in memory.
- Task structures are reused as tasks are done.


Performance of zpotrf using DTD PaRSEC with overlap, nb=128
Gflop/s
200
180
160
140
120
100
80
60
40
20
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Matrix Size
no_window, threads 20
overlap, threads 20
no_window, threads 16
overlap, threads, 16
no_window, threads 8
overlap, threads 8

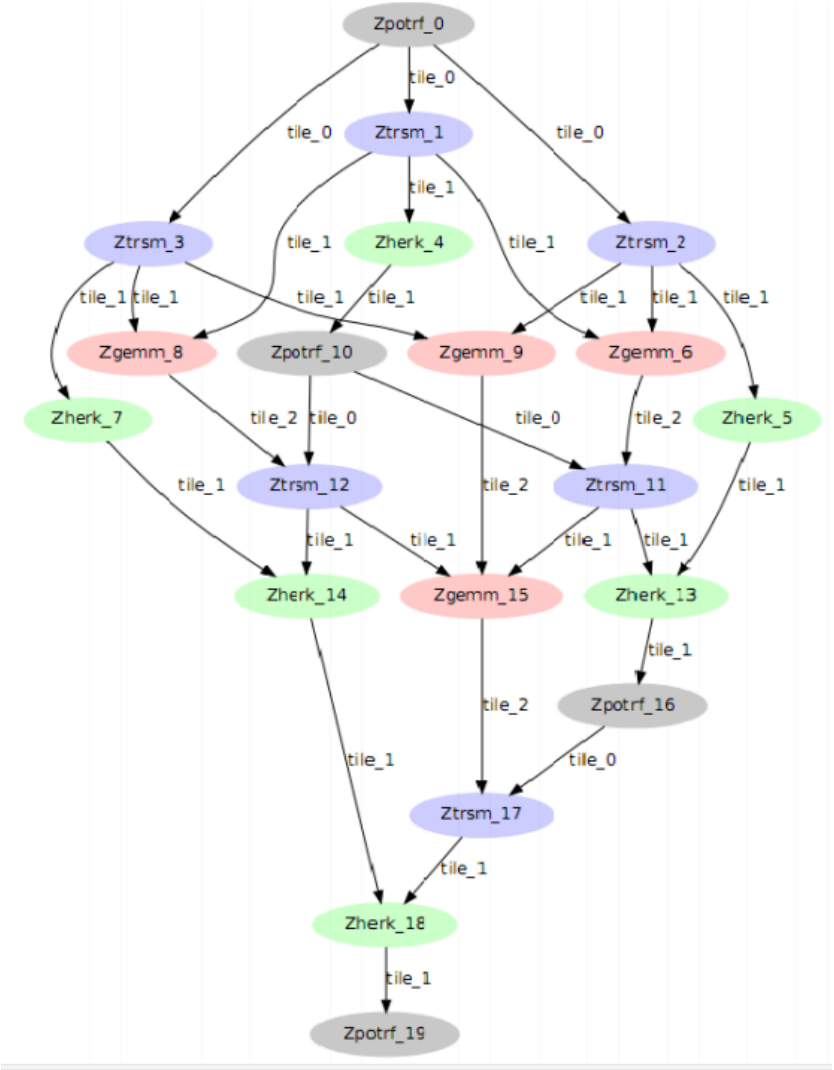

Performance of zgeqrf using DTD PaRSEC with overlap, nb=128
Gflop/s
Matrix Size
0
20
40
60
80
100
120
140
160
1000
2000
3000
4000
5000
6000
7000
8000
no_window, threads 20
overlap, threads 20
no_window, threads 16
overlap, threads, 16
no_window, threads 8
overlap, threads 8

Performance of zgetrf$_i$ncpiv using DTD PaRSEC with overlap on dancer, nb=



Development Tools: DAG Visualization How it was (DTD)

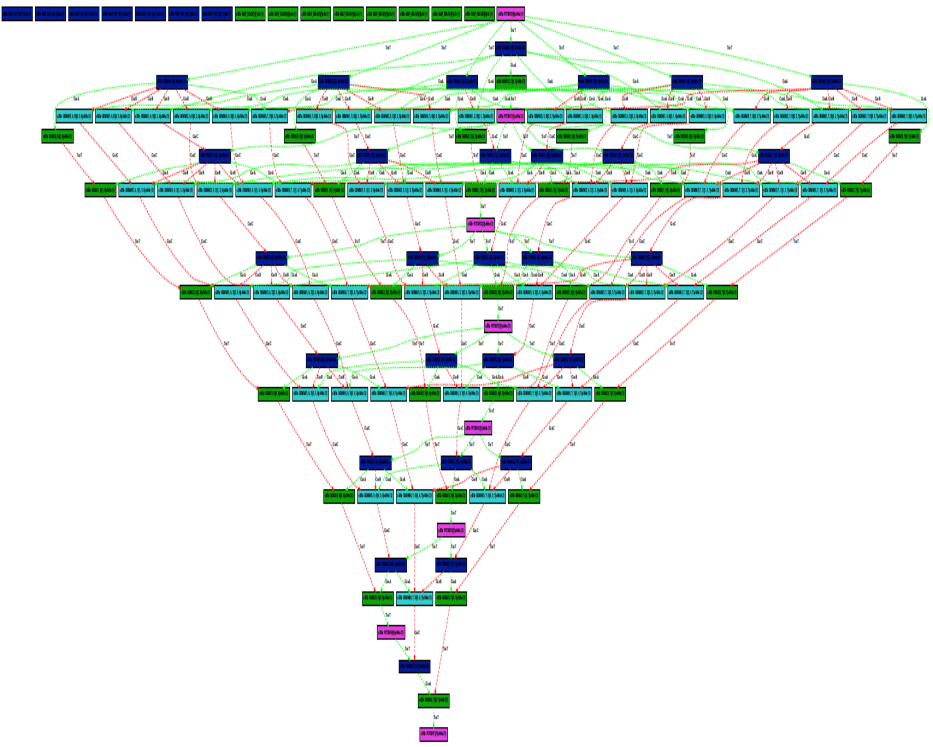
- Early development of DTD interface, provided rudimentary visualization help.
- Not integrated with the PaRSEC profiling system.



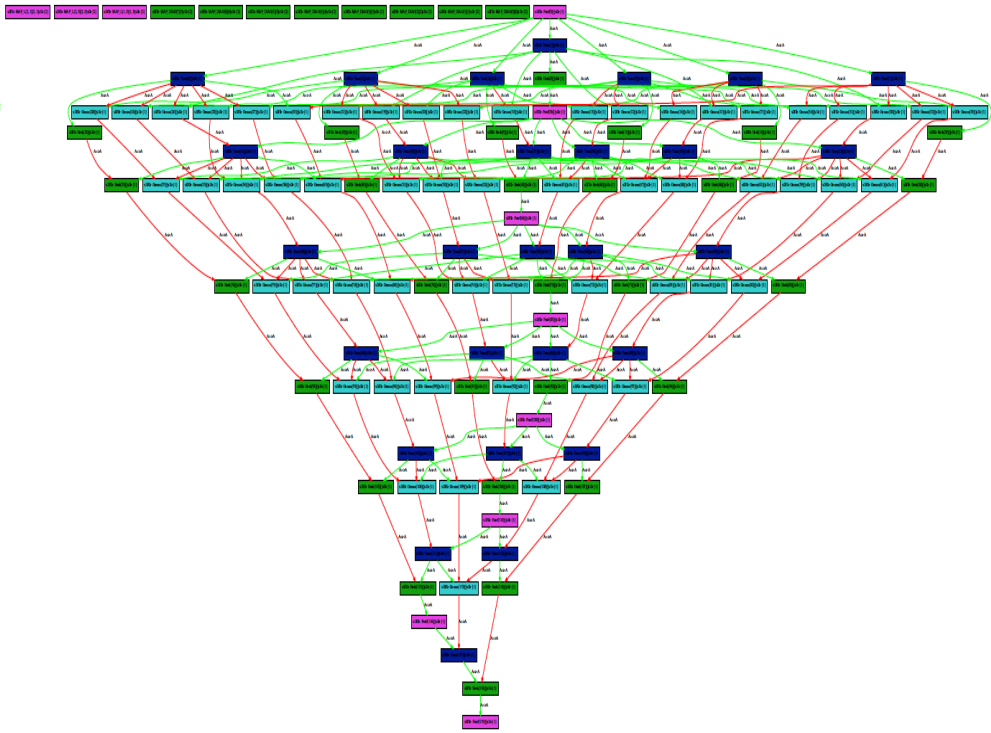
Development Tools: How it is

- All the profiling features available in PaRSEC are integrated in this interface.

PTG



DTD



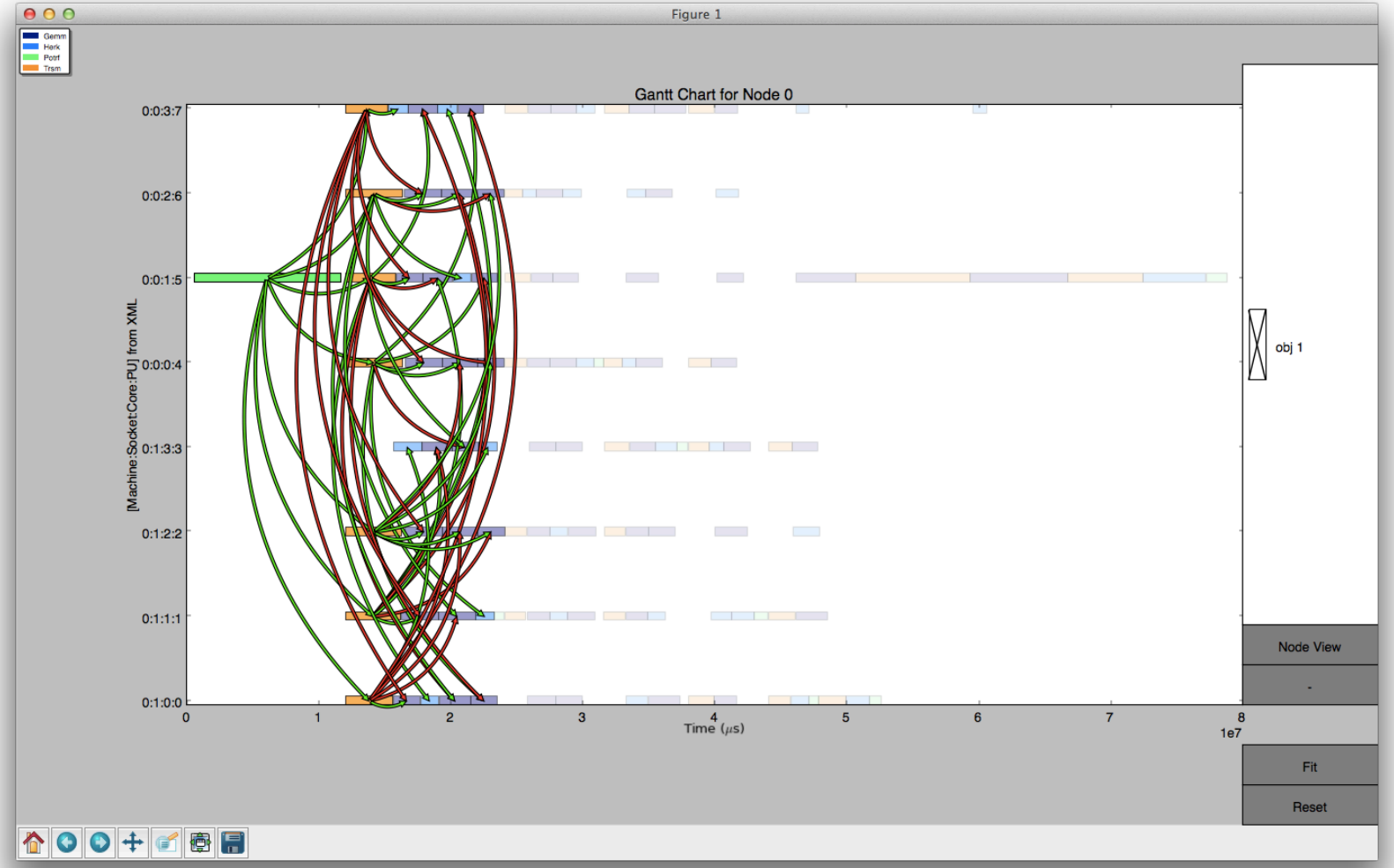


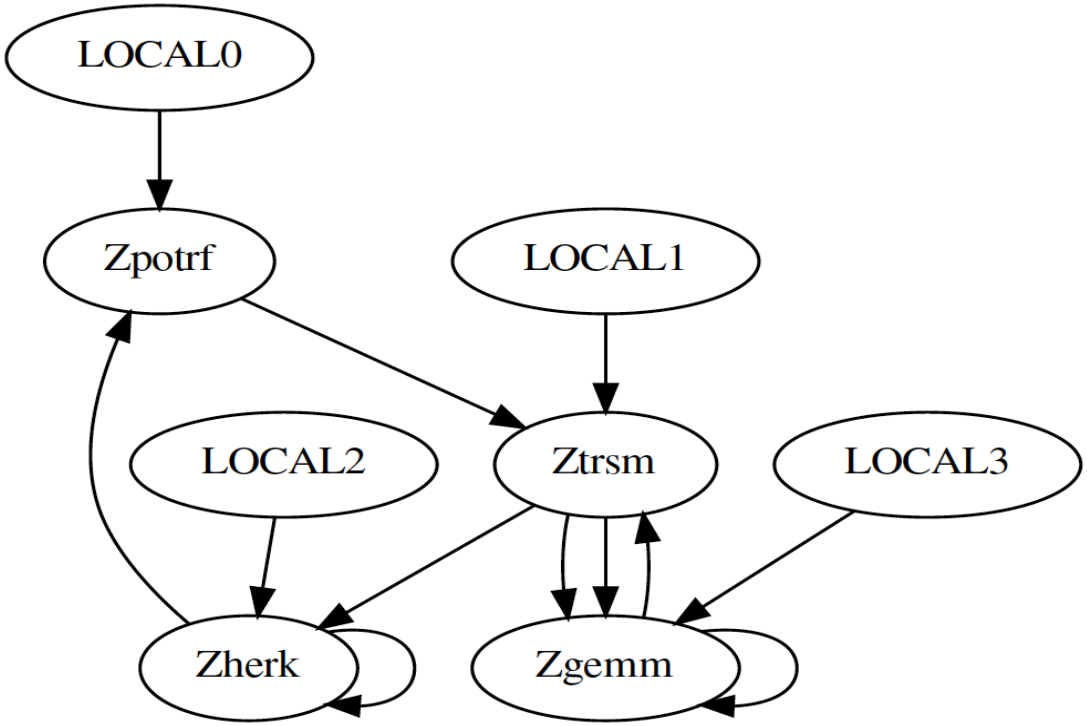
Figure 1
Gemm
Herk
Potrf
Trsm
Gantt Chart for Node 0
[Machine:Socket:Core:PU] from XML
0:0:3:7
0:0:2:6
0:0:1:5
0:0:0:4
0:1:3:3
0:1:2:2
0:1:1:1
0:1:0:0
0
1
2
3
4
5
6
7
8
1e7
Time (μs)
obj 1
Node View
-
Fit
Reset

Task Classes

- PaRSEC represents the DAG with Task classes
  - Tasks are instantiated when they become enabled and destroyed once they are executed.
  - E.g. POTRF (k) → Task class
    - Potrf (0), potrf (1) are task instances
- Previous work in DTD had only one task class and all the tasks were instances of that class.

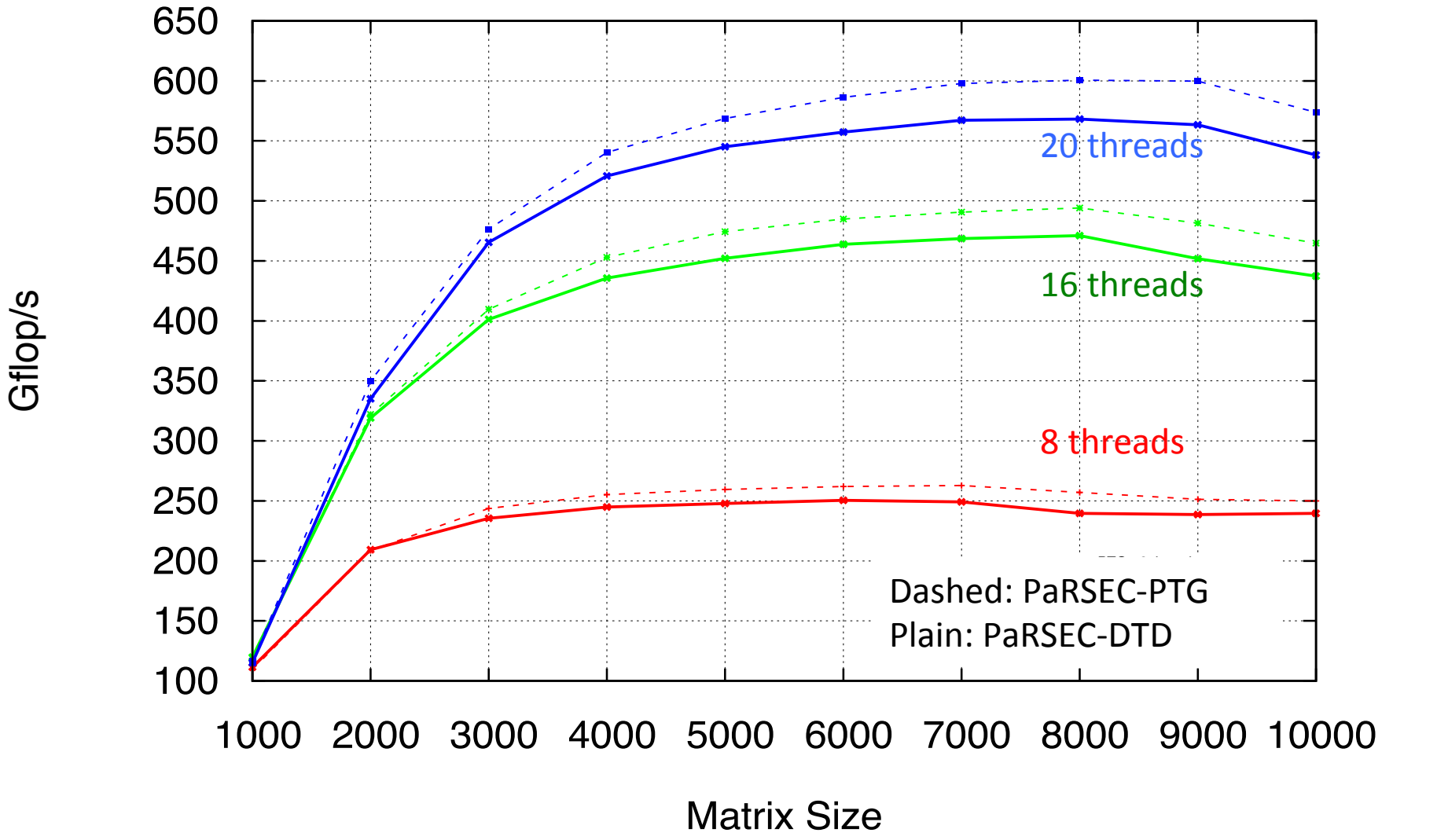
Task Classes

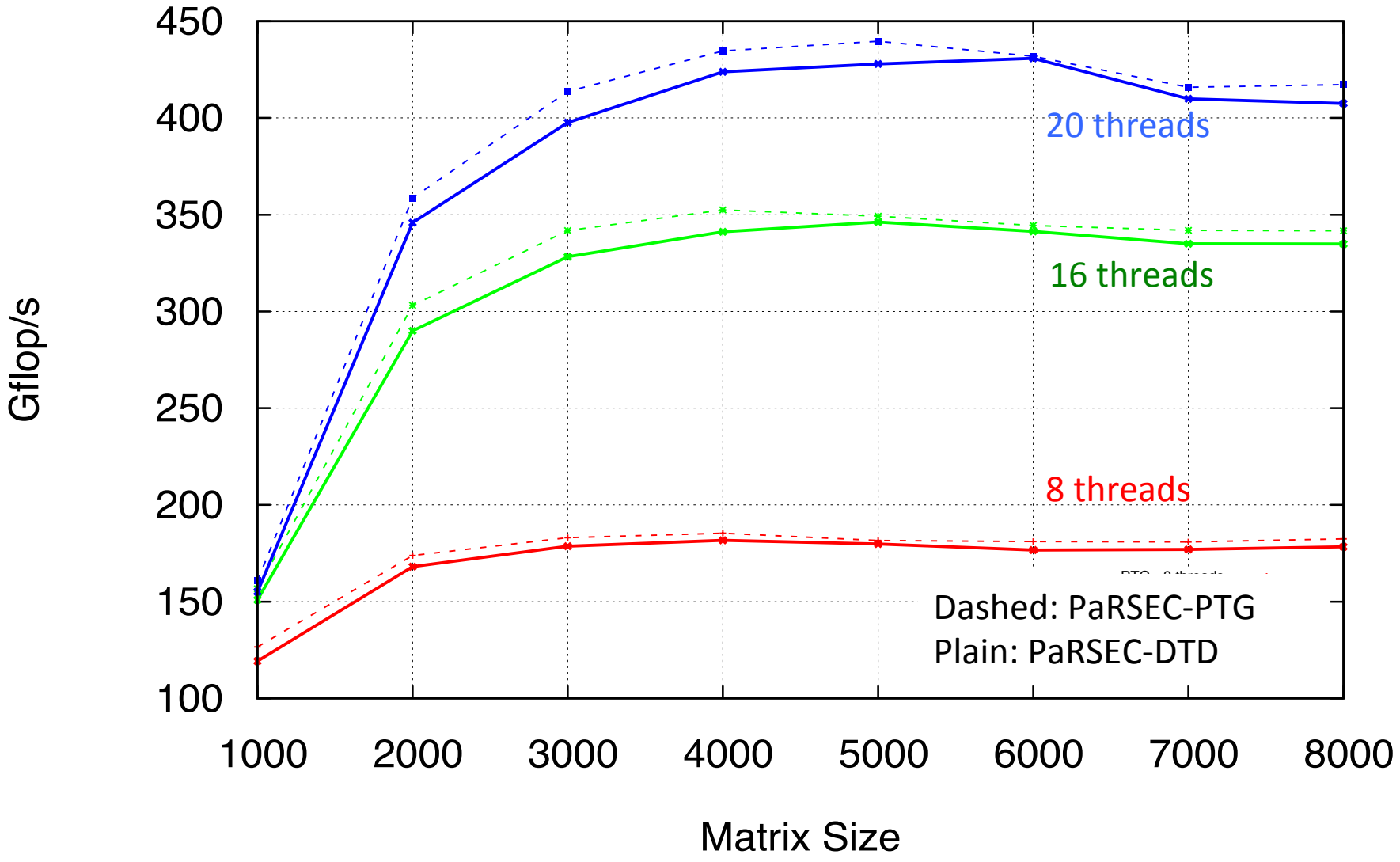
- Multiple task classes. One task class per function pointer.



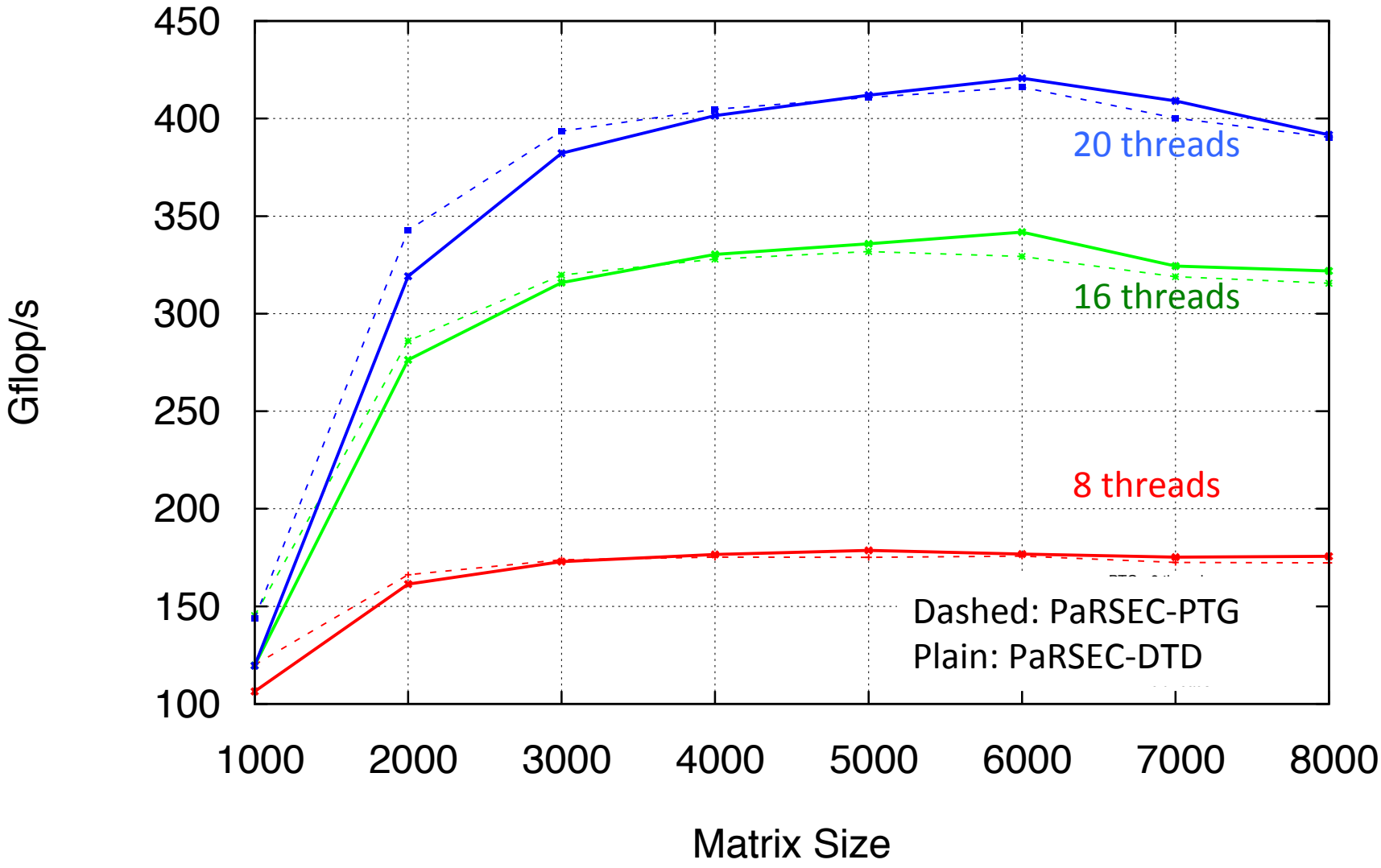
Validation

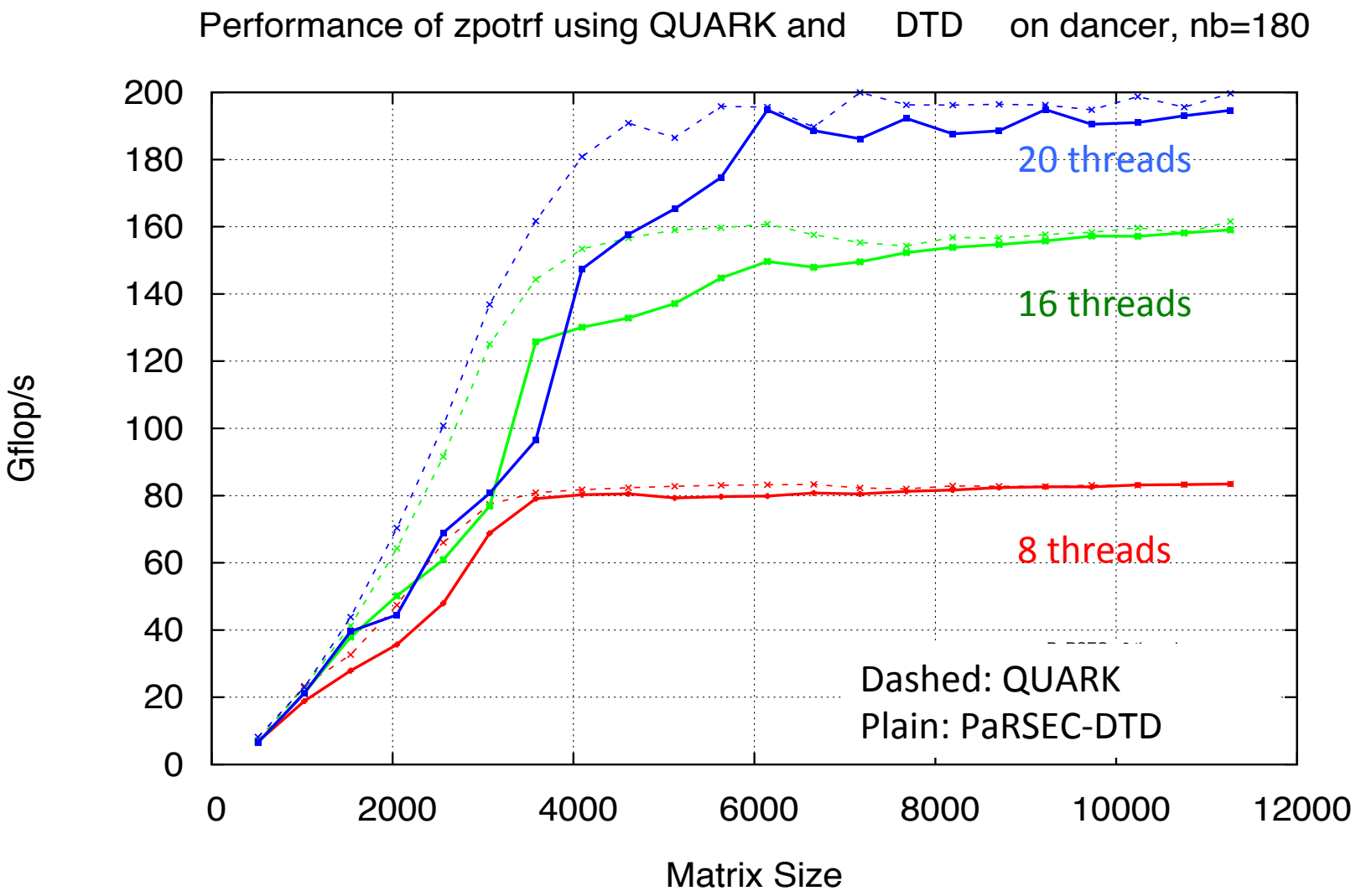
- Wrote three tests for PaRSEC
  - POTRF
  - GEQRF
  - GETRF_INCPIV
- Added PaRSEC DTD to MORSE
  - Have all the tests available in MORSE
- Instrumented a mechanism for PTG to insert task in DTD instead of executing them.
  - Have all the tests in PaRSEC

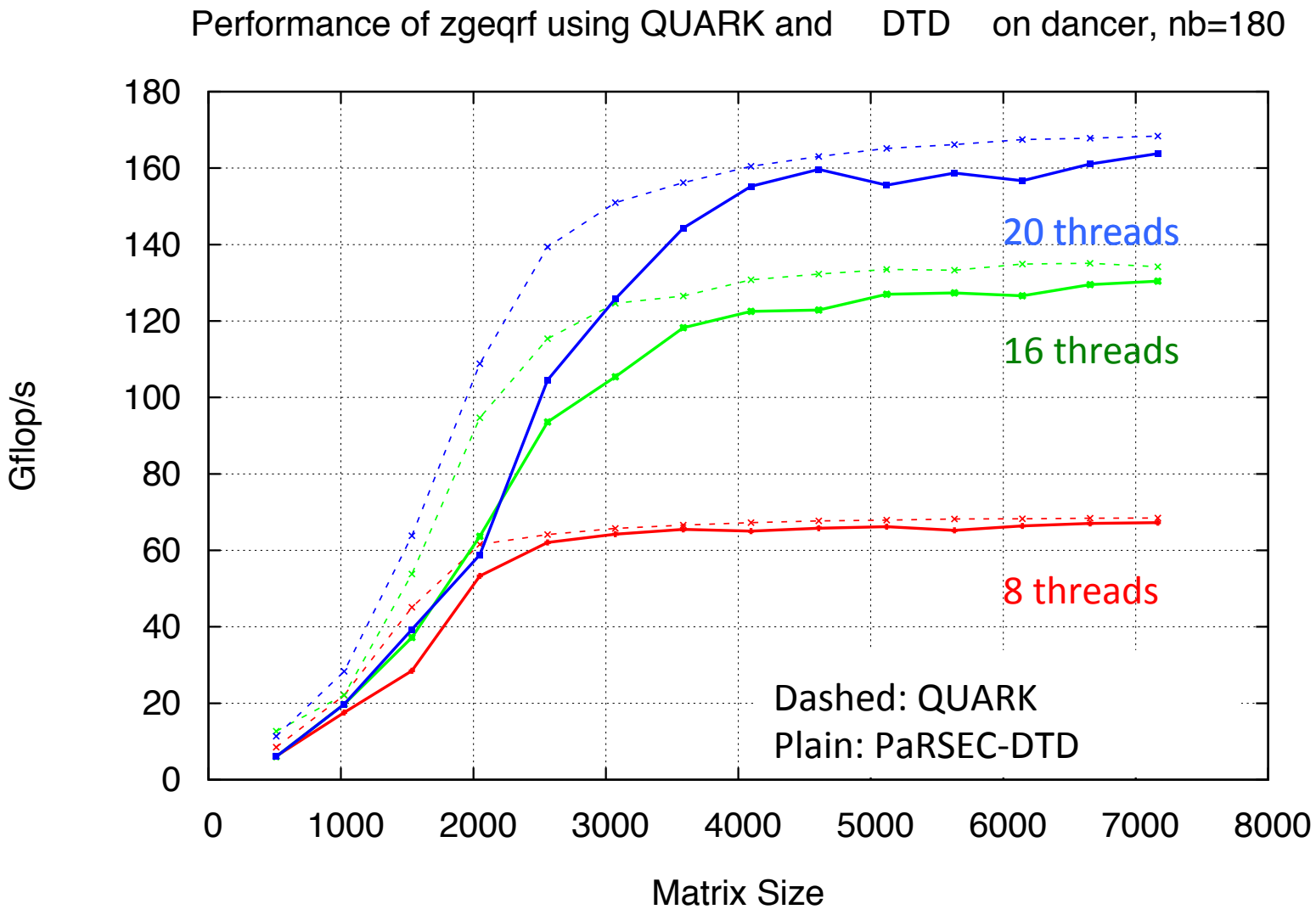

Performance of zpotrf using PTG and DTD of PaRSEC on dancer, nb=128
Gflop/s
650
600
550
500
450
400
350
300
250
200
150
100
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Matrix Size
20 threads
16 threads
8 threads
Dashed: PaRSEC-PTG
Plain: PaRSEC-DTD

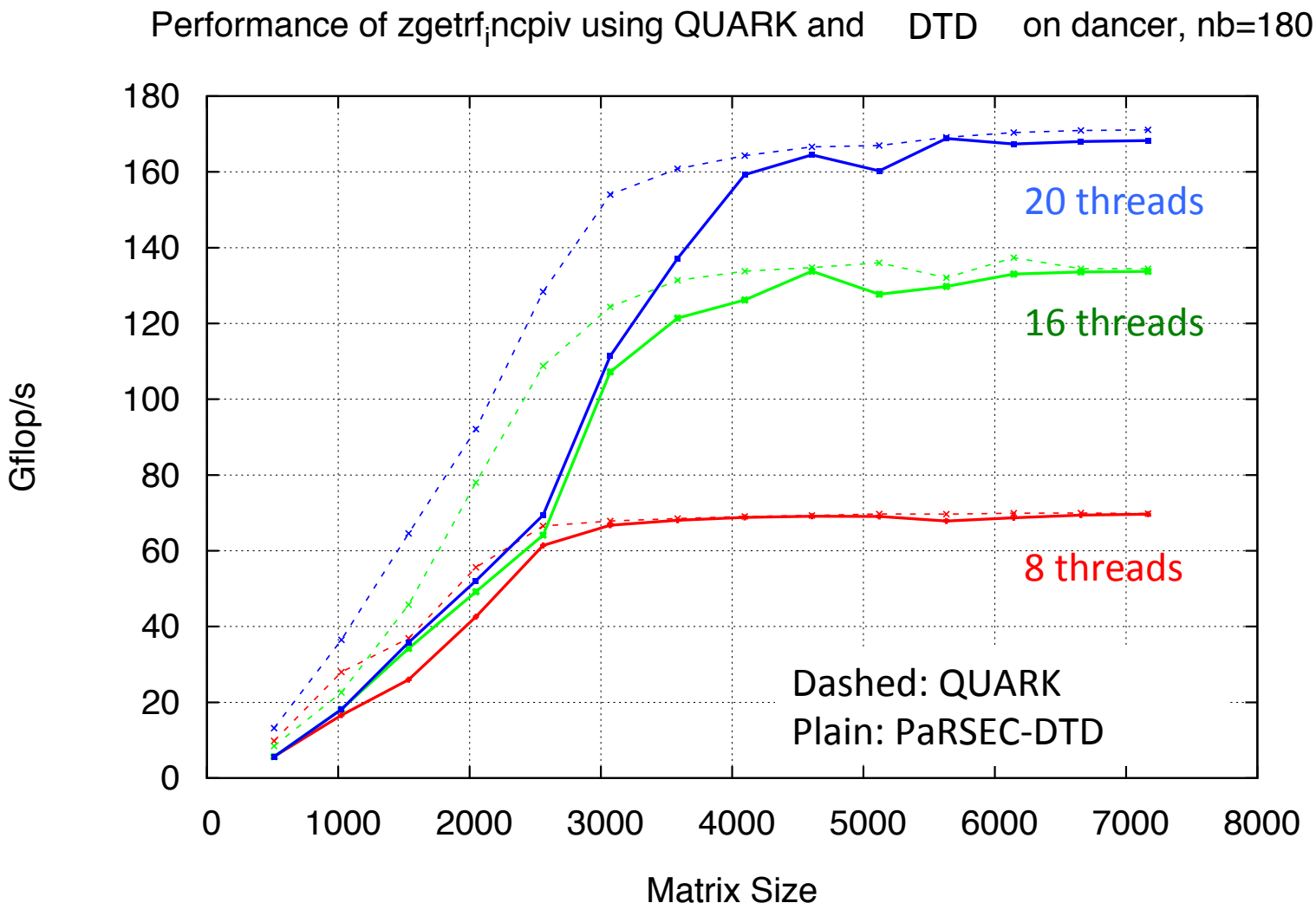

Performance of zgeqrf using PTG and DTD of PaRSEC on dancer, nb=128
Gflop/s
Matrix Size
450
400
350
300
250
200
150
100
1000
2000
3000
4000
5000
6000
7000
8000
20 threads
16 threads
8 threads
Dashed: PaRSEC-PTG
Plain: PaRSEC-DTD

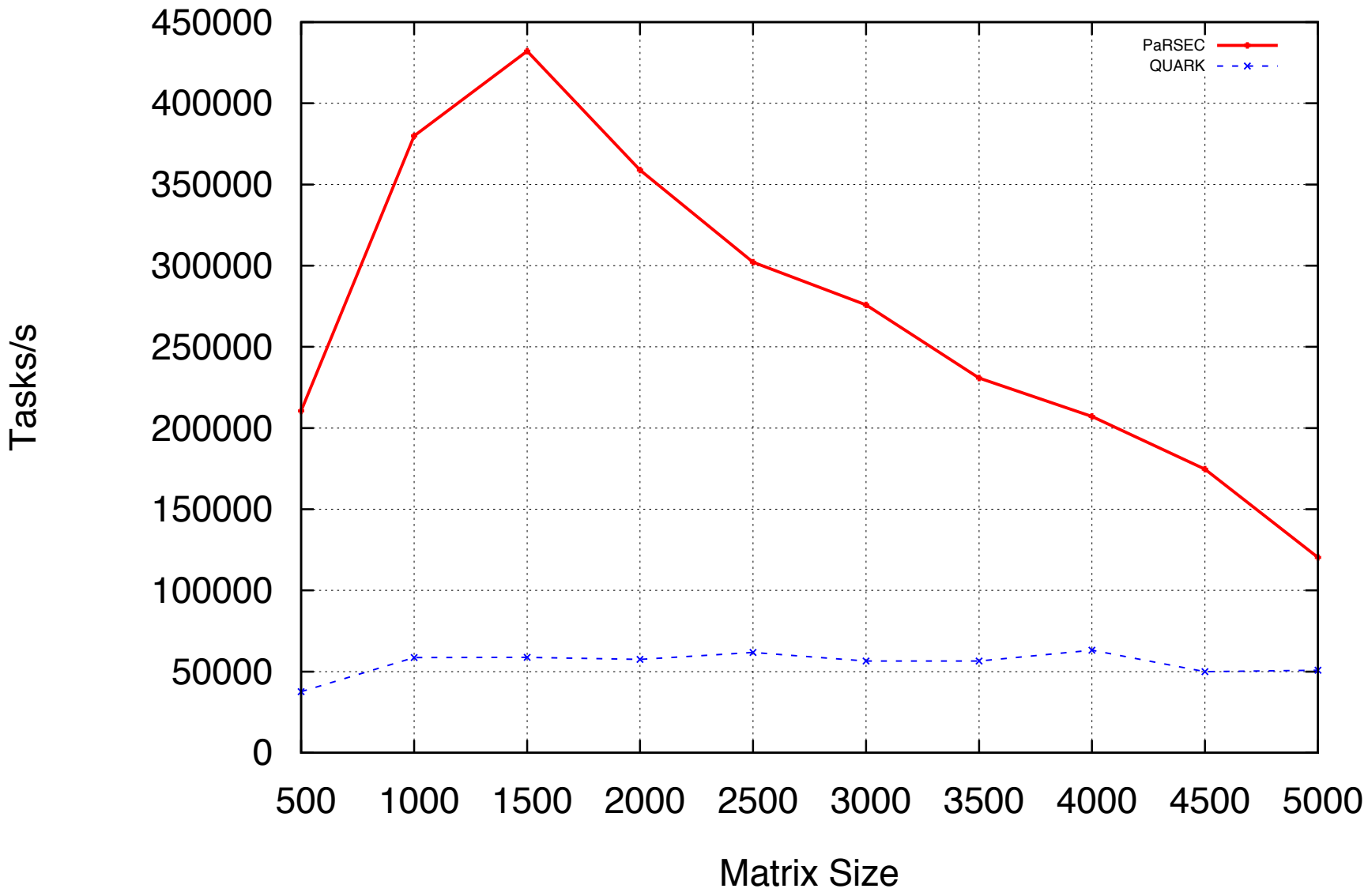
Performance of zgetrf_incpiv using PTG and DTD of PaRSEC on dancer, nb=128

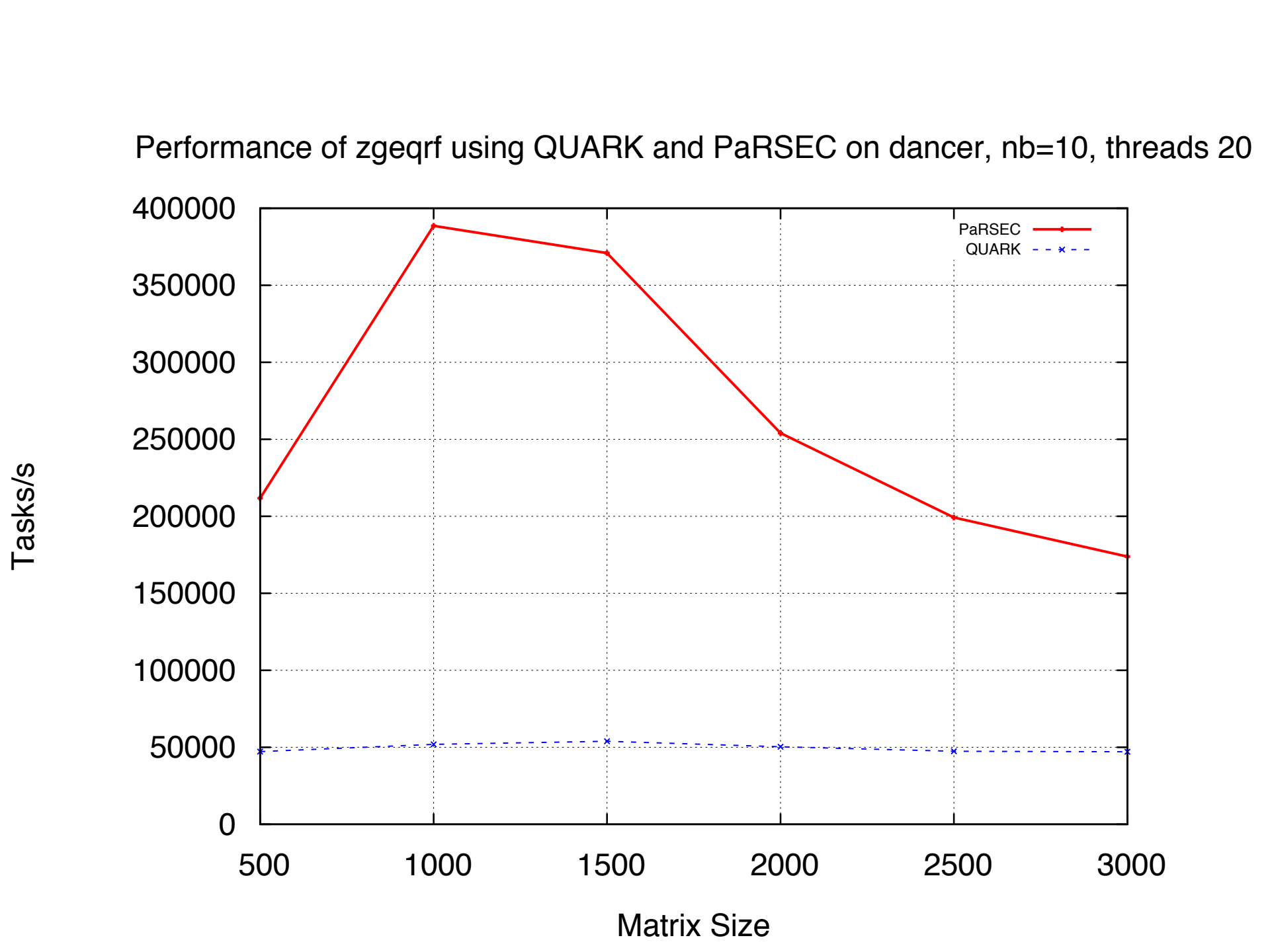


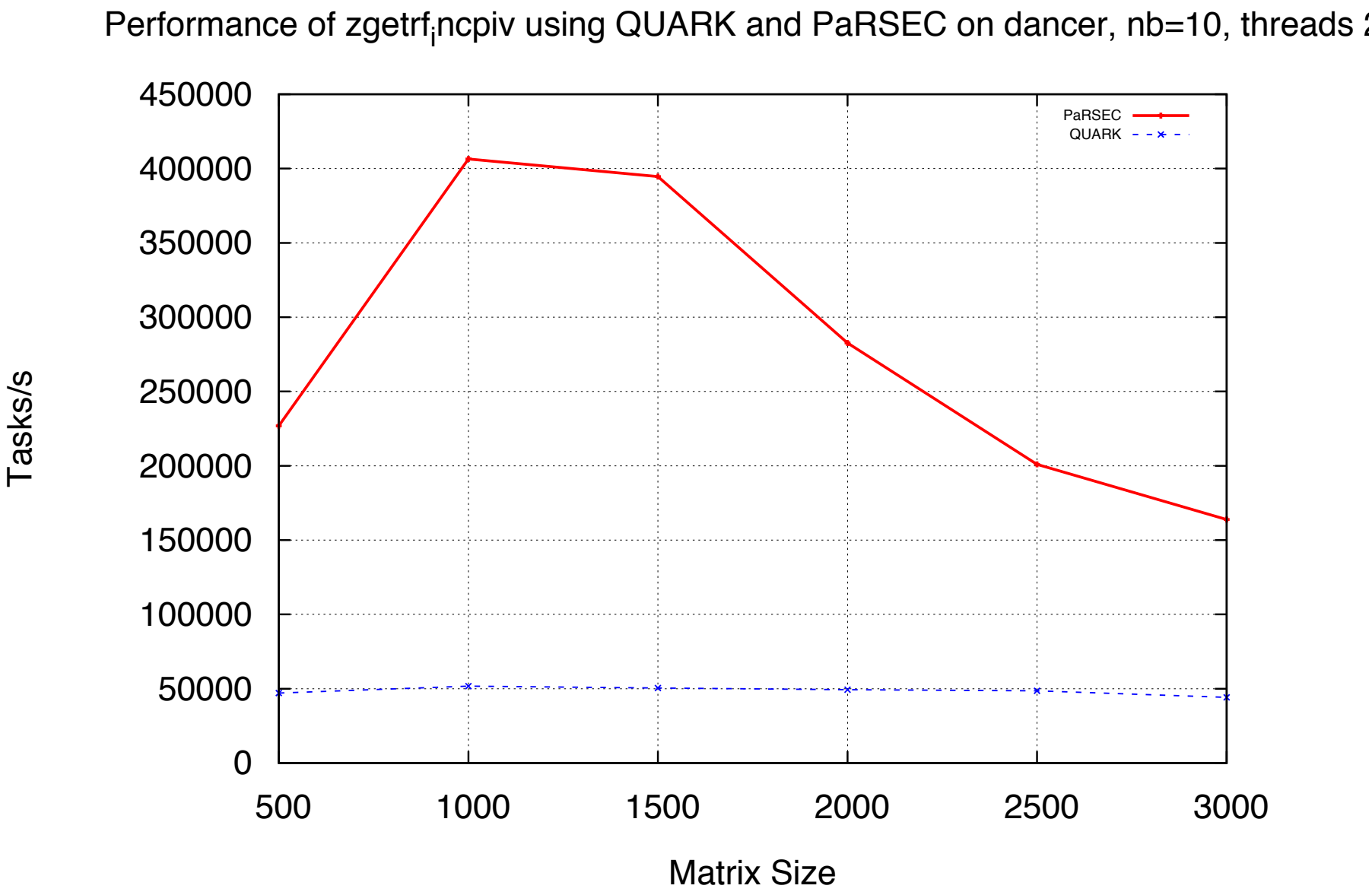

Performance of zpotrf using QUARK and DTD on dancer, nb=180
Gflop/s
Matrix Size
0
20
40
60
80
100
120
140
160
180
200
0
2000
4000
6000
8000
10000
12000
20 threads
16 threads
8 threads
Dashed: QUARK
Plain: PaRSEC-DTD


Performance of zgeqrf using QUARK and DTD on dancer, nb=180
Gflop/s
Matrix Size
0
20
40
60
80
100
120
140
160
180
0
1000
2000
3000
4000
5000
6000
7000
8000
20 threads
16 threads
8 threads
Dashed: QUARK
Plain: PaRSEC-DTD


Performance of zgetrf_incpiv using QUARK and DTD on dancer, nb=180
Gflop/s
Matrix Size
0
20
40
60
80
100
120
140
160
180
0
1000
2000
3000
4000
5000
6000
7000
8000
20 threads
16 threads
8 threads
Dashed: QUARK
Plain: PaRSEC-DTD


Performance of zpotrf using QUARK and PaRSEC on dancer, nb=10, threads 20
Tasks/s
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Matrix Size
PaRSEC
QUARK


Performance of zgeqrf using QUARK and PaRSEC on dancer, nb=10, threads 20
Tasks/s
400000
350000
300000
250000
200000
150000
100000
50000
0
500
1000
1500
2000
2500
3000
Matrix Size
PaRSEC
QUARK


Performance of zgetrf_incpiv using QUARK and PaRSEC on dancer, nb=10, threads 2
Tasks/s
Matrix Size
PaRSEC
QUARK
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
500
1000
1500
2000
2500
3000

Future Work

- Support for GPU
- Going Distributed