WALLPAPER BY ronnie
toward the exa
DPLASMA
MAGMA
PLASMA
Happy Halloween
2013
Azzam Haidar
www.ronniesaini.com

Toward fast DLA solver

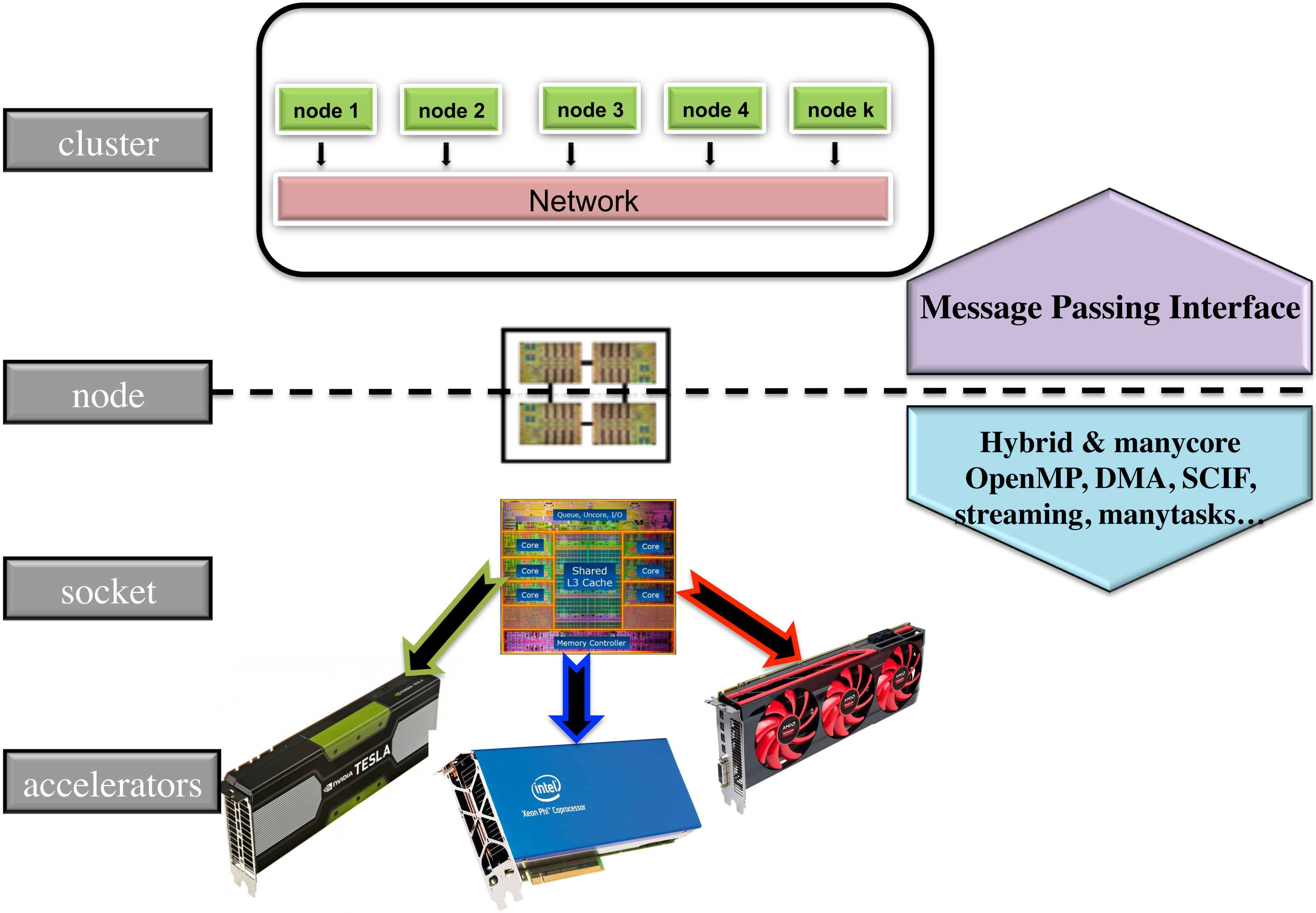
- The roadmap toward efficient Dense Linear Algebra software

- Current trends and what to expect for long term?

High Performance Computing : current trends

- New Constraints, Moore’s Law reinterpreted
  - Clock rate growth has ended, #cores per chip doubles instead of clock frequency,
  - Architecture and programming model are reinterpreted and changing,
  - Need to deal with systems with millions of concurrent threads,
  - Need to deal with inter-chip parallelism as well as intra-chip parallelism,
  - Impact on Software: We will need to rethink and redesign our algorithms to be adapted to the new tech. Similar challenge as in the 1990-1995 transition to clusters and MPI.

High Performance Computing : current trends



cluster

node 1

node 2

node 3

node 4

node k

Network

Message Passing Interface

node

Hybrid & manycore
OpenMP, DMA, SCIF,
streaming, manytasks…

socket

accelerators

High Performance Computing : current development

We present here a feasibility design study, the idea is to target the new high-end technologies.

Our goals:

- is to develop a programming model that raises the level of abstraction above the hardware and its accompanying software stack to offer uniform approach for algorithmic development.
- also our idea is to try to use such prototype to influence the HPC industry designs,
- or to provide the user/engine some guides and answers about the performance of his application.
- is to consider both, the higher ratio of execution and the hierarchical memory model of the new emerging accelerators and coprocessors.

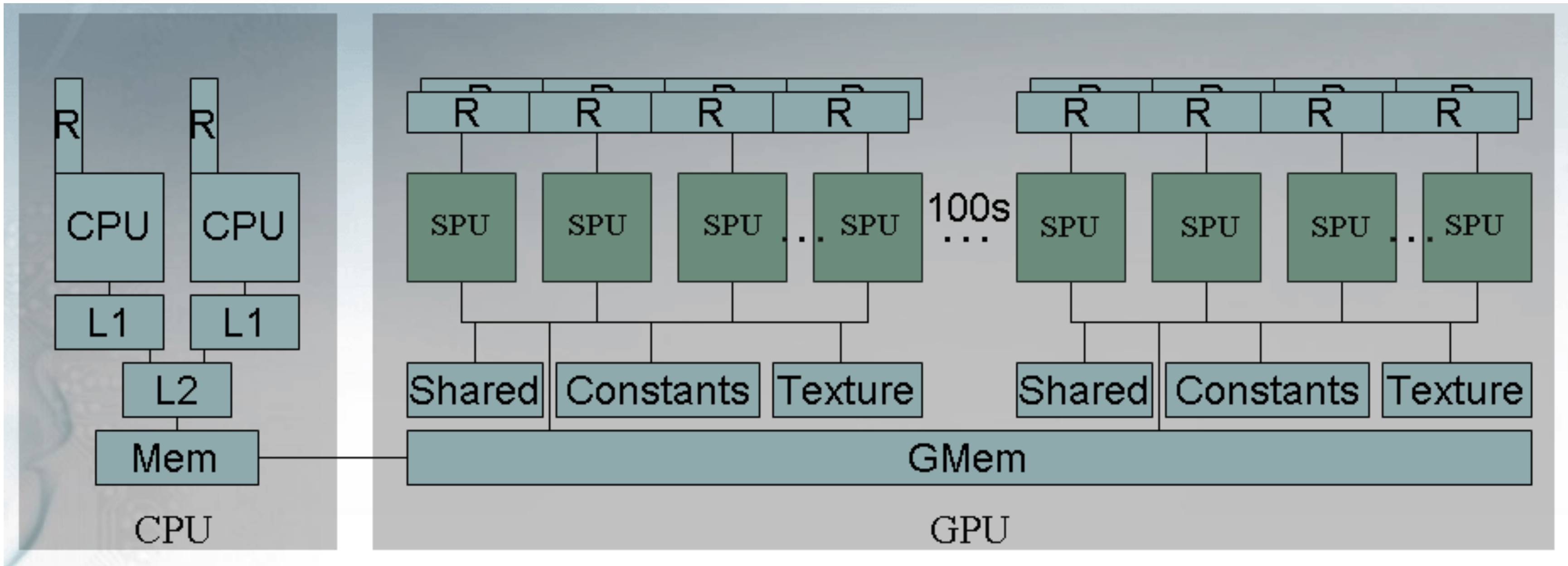
High Performance Computing : current development

We present here a feasibility design study, the idea is to target the new high-end technologies.

Key features:

- processing unit capability (CPUs, GPUs, Xeon Phi),
- the memory access,
- and the communication cost.

High Performance Computing : current development



As with CPUs, the access time to the device memory for accelerators is slow compared to peak performance:

- CPUs try to improve the effect of the long memory latency and bandwidth by using hierarchical caches.
- accelerators use multithreading operations that access large data sets. By running thousands of fast-switching light threads large memory latency can be masked.

High Performance Computing : current development

we develop a strategy :

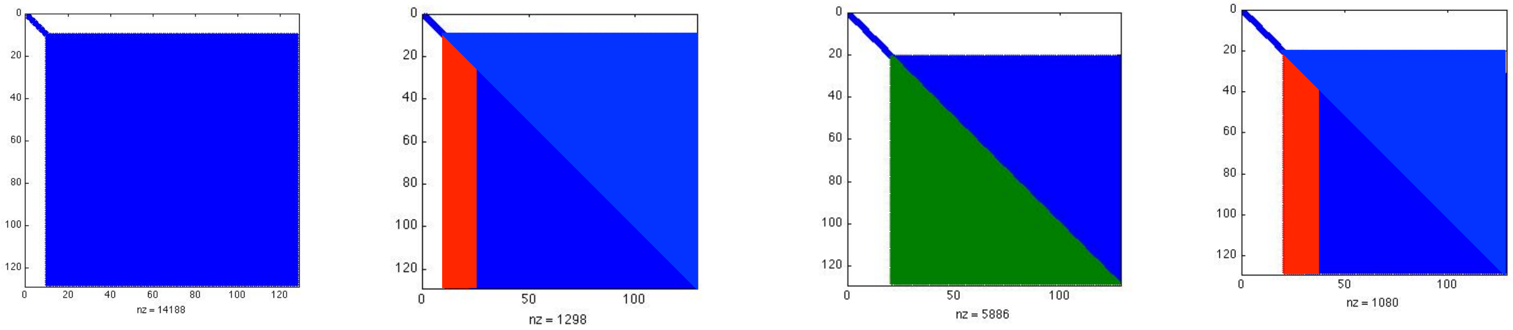
- that prioritizes the data-intensive operations to be executed by the accelerator

- that keep the memory-bound ones for the CPUs since the hierarchical caches with out-of-order superscalar scheduling are more appropriate to handle it.

- Moreover, in order to keep the accelerator busy, we redesign the kernels and propose dynamically guided data distribution to exploit enough parallelism to keep the accelerators and processors busy.

High Performance Computing : current development

1. Standard hybrid CPU-GPU implementation

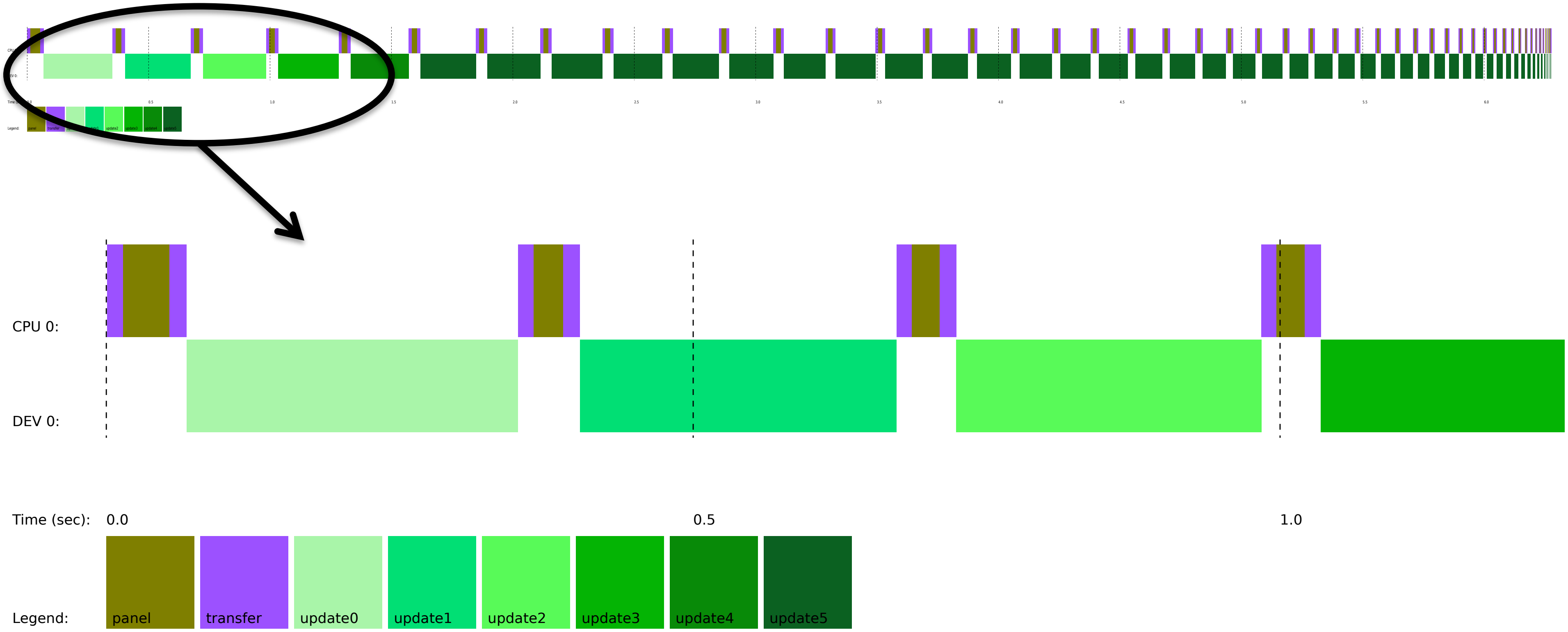


factor panel k then update ➔ factor panel k+1

Algorithm 1: Two-phase implementation of a one-sided factorization.

```
for $P_i \in \{P_1, P_2, \dots, P_n\}$ do
    CPU:
    Receive Panel($P_i$)
    PanelFactorize($P_i$)
    Send Panel($P_i$)
    GPU:
    TrailingMatrixUpdate($A^{(i)}$)
```

High Performance Computing : current development



Standard implementation without lookahead:

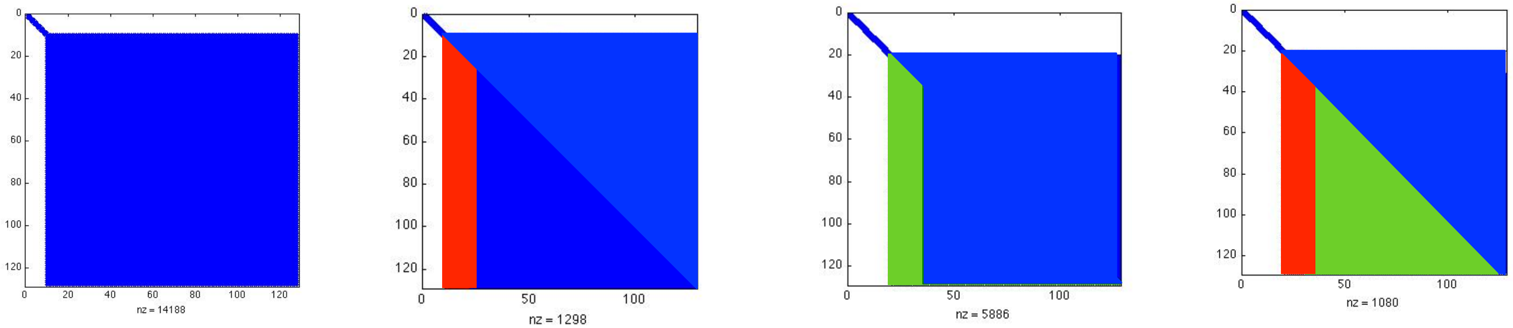
➢Execution trace of the Cholesky factorization on a single socket CPU (Sandy Bridge) and a K20c GPU.

➢We see that the computation on the CPU (e.g., the panel factorization) is not overlapped with the computation on the GPU.

➢The algorithm looks like sequential, the only advantage is that the data extensive operations are accelerated by the GPU.

High Performance Computing : current development

2. Introducing a lookahead panel to overlap CPU and GPU



factor panel k

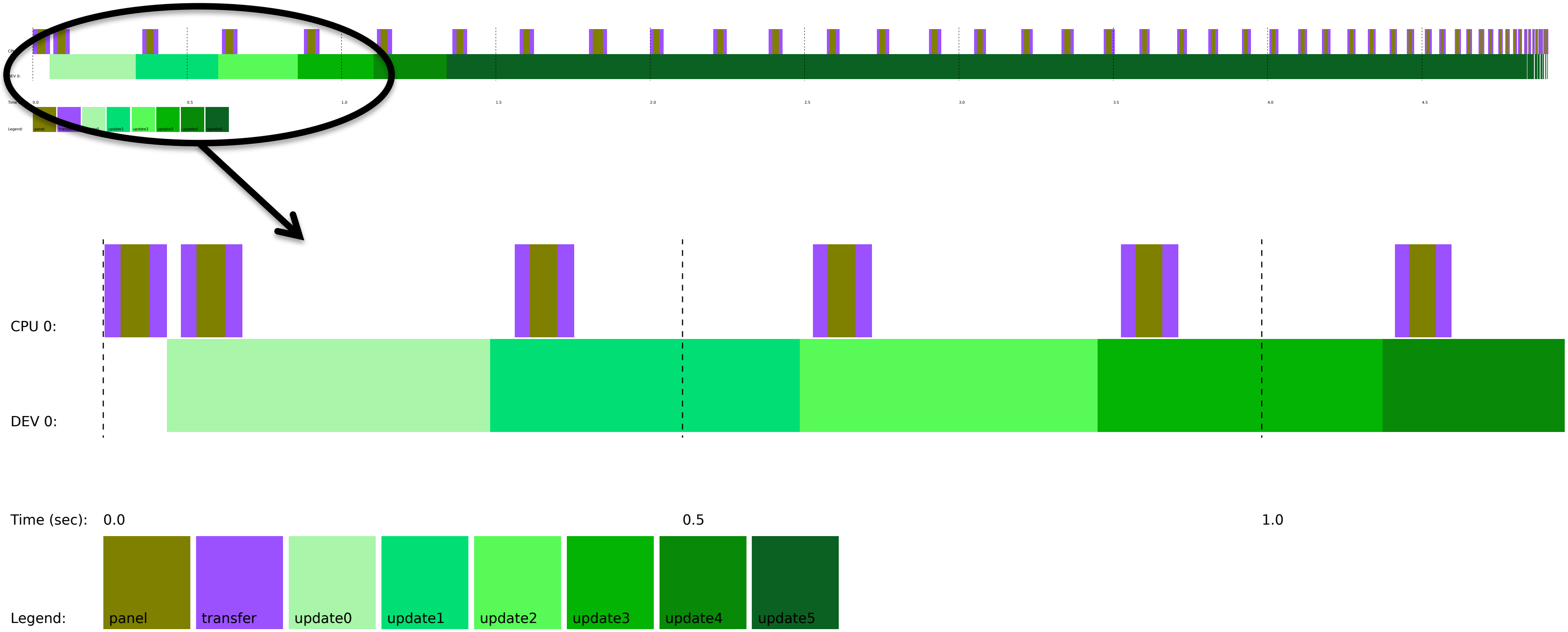
then update next panel ➔

factor panel k+1 continue update k

Algorithm 2: Two-phase implementation with a split update and explicit communication.

```
for $P_i \in \{P_1, P_2, \ldots, P_n\}$ do
    CPU:
    Receive Panel($P_i$)
    PanelFactorize($P_i$)
    Send Panel($P_i$)
    GPU:
    NextPanelUpdate(lookahead $P_{(i+1)}$) $\rightarrow$ goto CPU
    TrailingMatrixUpdate($A^{(i)}$)
```

High Performance Computing : current development



New implementation with lookahead:

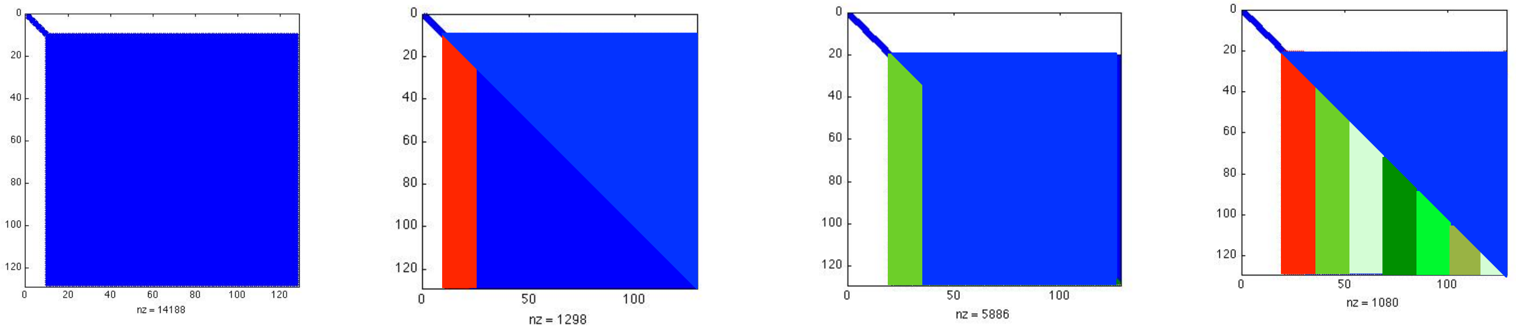
➢Execution trace of the Cholesky factorization on a single socket CPU (Sandy Bridge) and a K20c GPU.

➢We see that the memory-bound kernel (e.g., the panel factorization) has been allocated to the CPU while the compute-bound kernel (e.g., the update performed by DSYRK) has been allocated to the accelerator.

➢the advantage of such strategy is not only to hide the data transfer cost between the CPU and GPU but also to keep the GPU busy all the way until the end of execution.

High Performance Computing : current development

3. Prioritizing critical path to provide more parallelism if needed



factor panel k

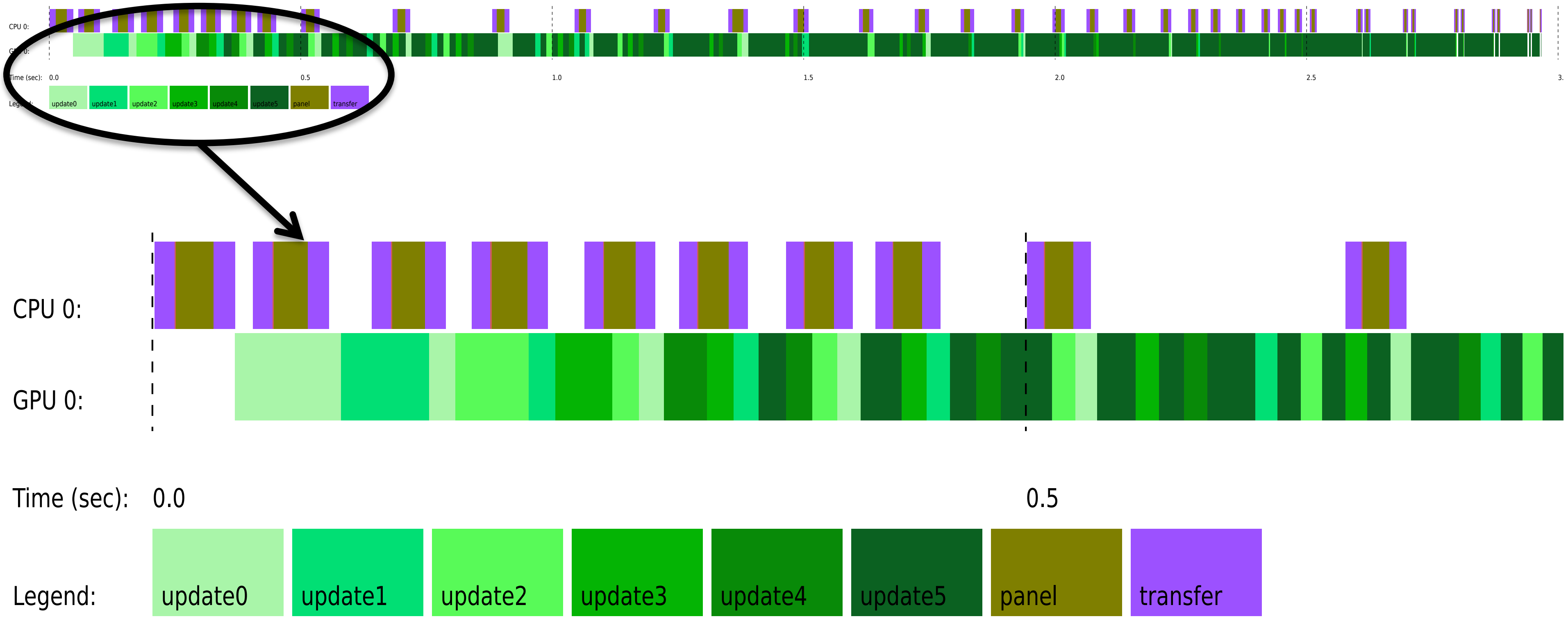
then update next panel

➔ factor panel k+1 continue update k

Algorithm 3: Two-phase implementation with a split update prioritizing critical path.

```
for $P_i \in \{P_1, P_2, \ldots, P_n\}$ do
    CPU:
    Receive Panel($P_i$)
    PanelFactorize($P_i$)
    Send Panel($P_i$)
    GPU:
    for $j \in \{P_{i+1}, P_{i+2}, \ldots, P_n\}$ do
        MatrixUpdate of block j($P_{(j)}$) with priority $p - j$
```

High Performance Computing : current development



Prioritize the critical path :

➢the panel factorization can be executed earlier. This will increase the lookahead depth that the algorithm exposes, increasing parallelism, so that there are more update tasks available to be executed by the device resources.

➢This options has advantage when a lot of parallelism is needed especially for small sizes.

High Performance Computing : current development

4. Introducing Resource Capability Weight

- **Room for improvements:**

  - Clearly, we cannot balance the load, if we treat CPU and GPU as peers and assign them equivalent amount of work this naïve strategy would cause the accelerator to be substantially idle.

  - On a multi-heterogeneous devices platform, if the data is equally distributed (for example 1D-2D block cyclic), the execution flow will be bound by the performance of the slowest machine and thus resources are poorly exploited resulting in an unsatisfactory performance.

- **Objectives:**

  - improve further the implementation by fully utilizing all the available resources, particularly exploiting the idle time of the CPUs.

  - Define a Hardware-Guide Data Distribution especially when dealing with heterogeneous accelerators devices (GPUs, Xeon Phi). the data is either distributed or redistributed in an automatic fashion so that each device gets the appropriate volume of data to match its capabilities.

High Performance Computing : current development

4. Introducing Resource Capability Weight

- **Define a resource Capability weight:**

  - The resource capability-weights (**CW**) for a task is then the ratio of the total cost of the task on one resource versus another resource.

  - This cost is based on the communication cost (if the data has to be moved) and on the type of computation (memory-bound or compute-bound) performed by the task.

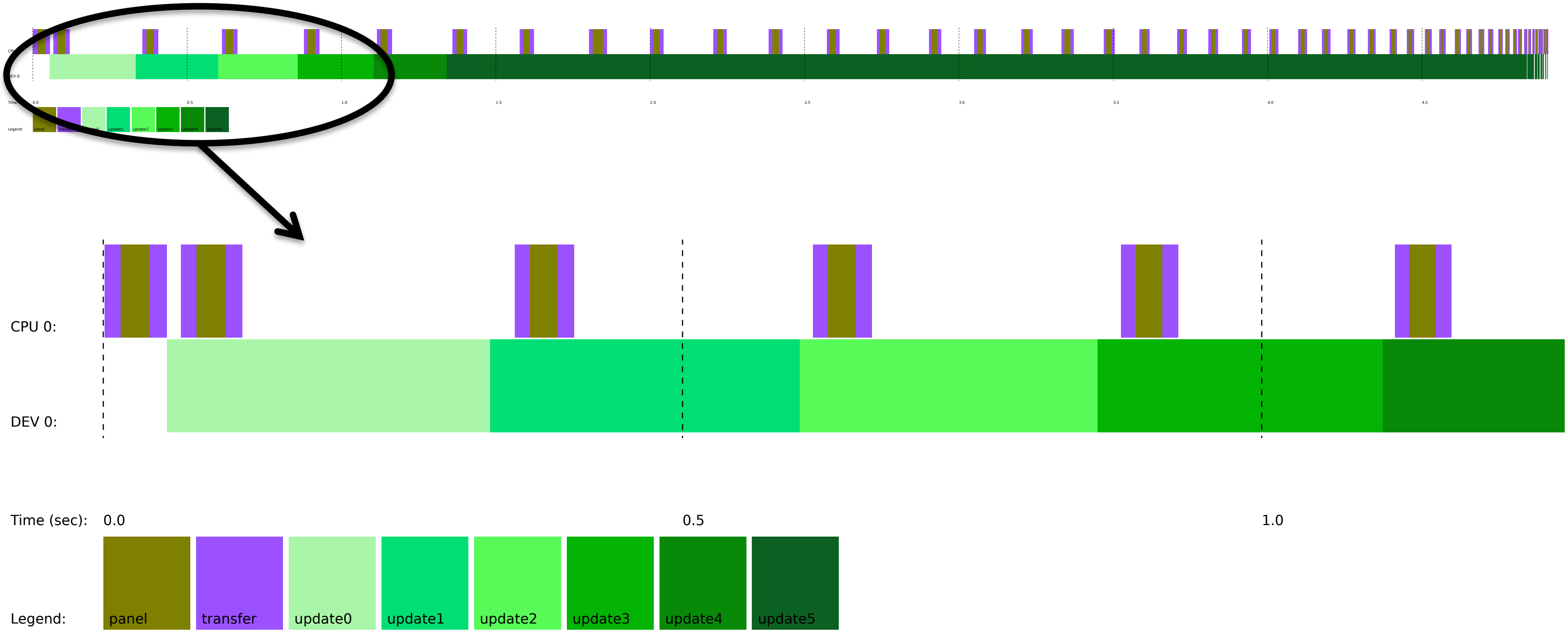
    - $comm = \dfrac{n \times n}{bandwidth}$

    - $compute = n^k \times \alpha_{ik}$

    - where k is the type of an operation (BLAS-1, BLAS-2, BLAS-3) and for each device ***i*** and for each kernel of type ***k***, we maintains an $\boldsymbol{\alpha_{ik}}$ parameter which corresponds to the effective performance rate that can be achieved on that device. $\boldsymbol{\alpha_{ik}}$ can be given either by the developer during the implementation, or estimated according to the volume of data and the elapsed time of a kernel by the QUARK engine at runtime.

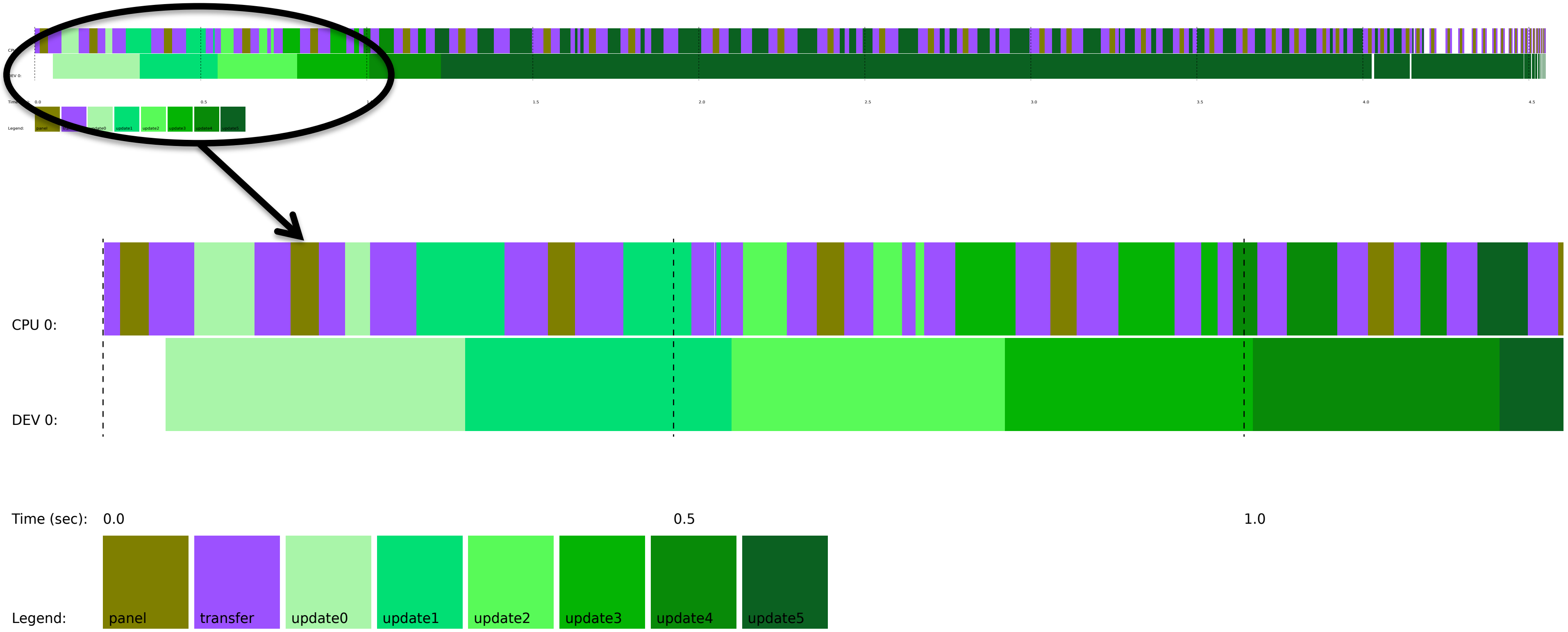
  - A task can be executed on a device with ***low CW*** *if and only if* its cost is less than the sum of costs of the queued tasks on devices with **higher CW**.

High Performance Computing : current development



No Capability Weight

High Performance Computing : current development

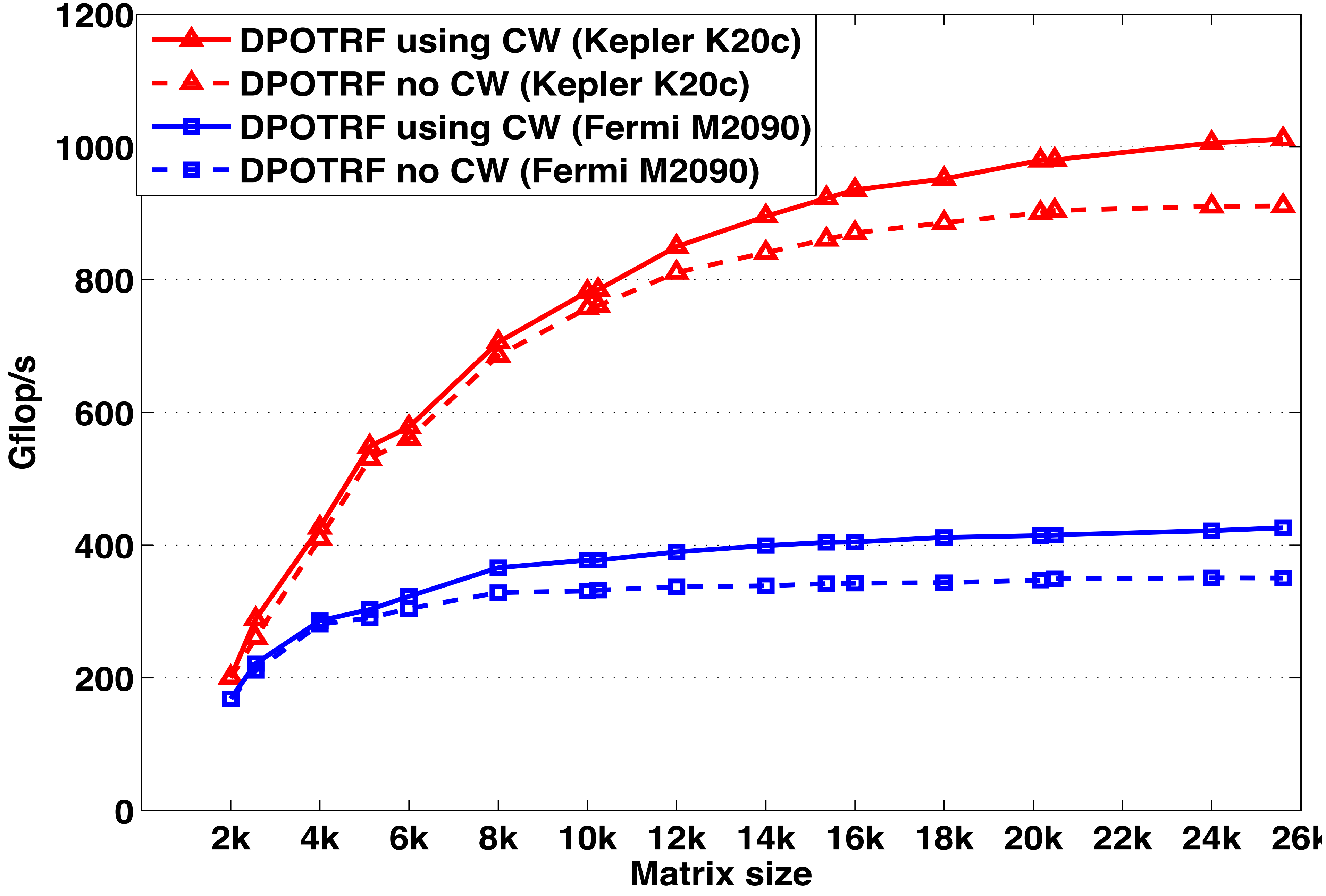


Resource Capability Weight :

➢ the advantage of such strategy is to keep all resources busy all the way until the end of execution.

➢ Careful management of the capability-weights ensures that the CPU does not take any work that would cause a delay to the GPU, since that would negatively affect the performance.

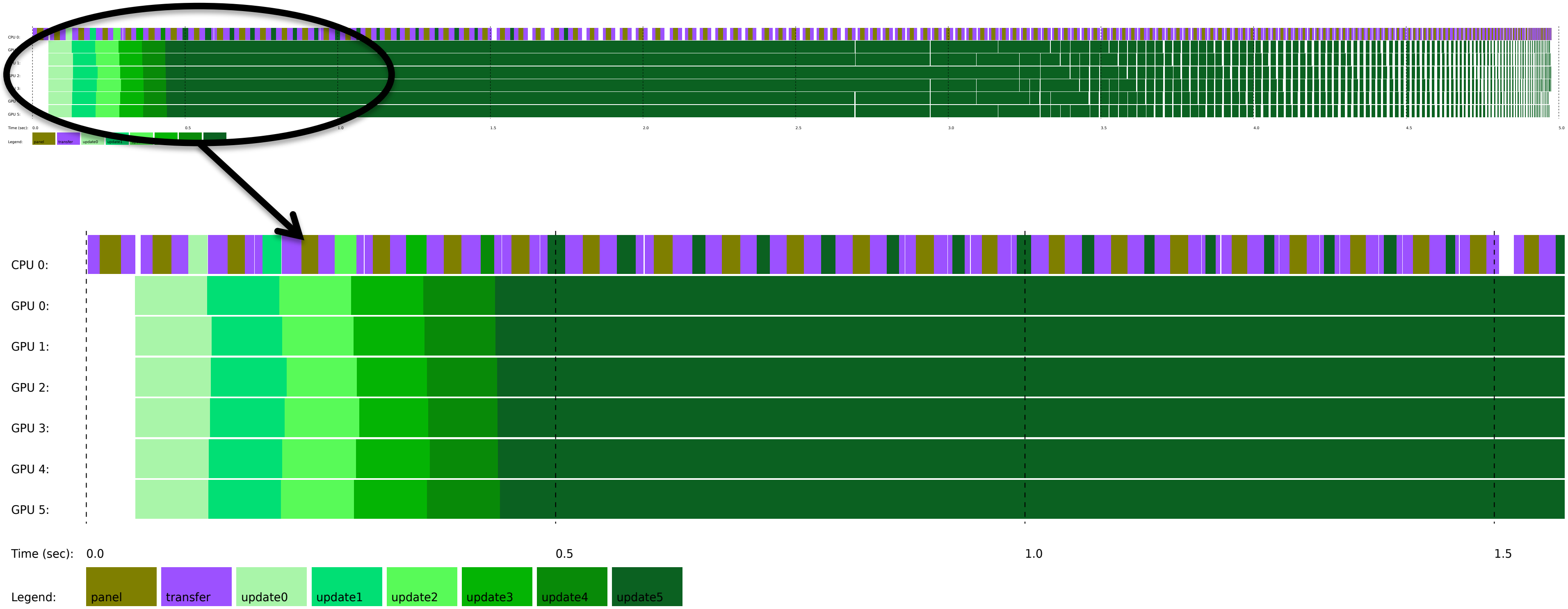
High Performance Computing : current development



High Performance Computing : current development

Scalability and performance of such implementation

High Performance Computing : current development



Scalability and efficiency :

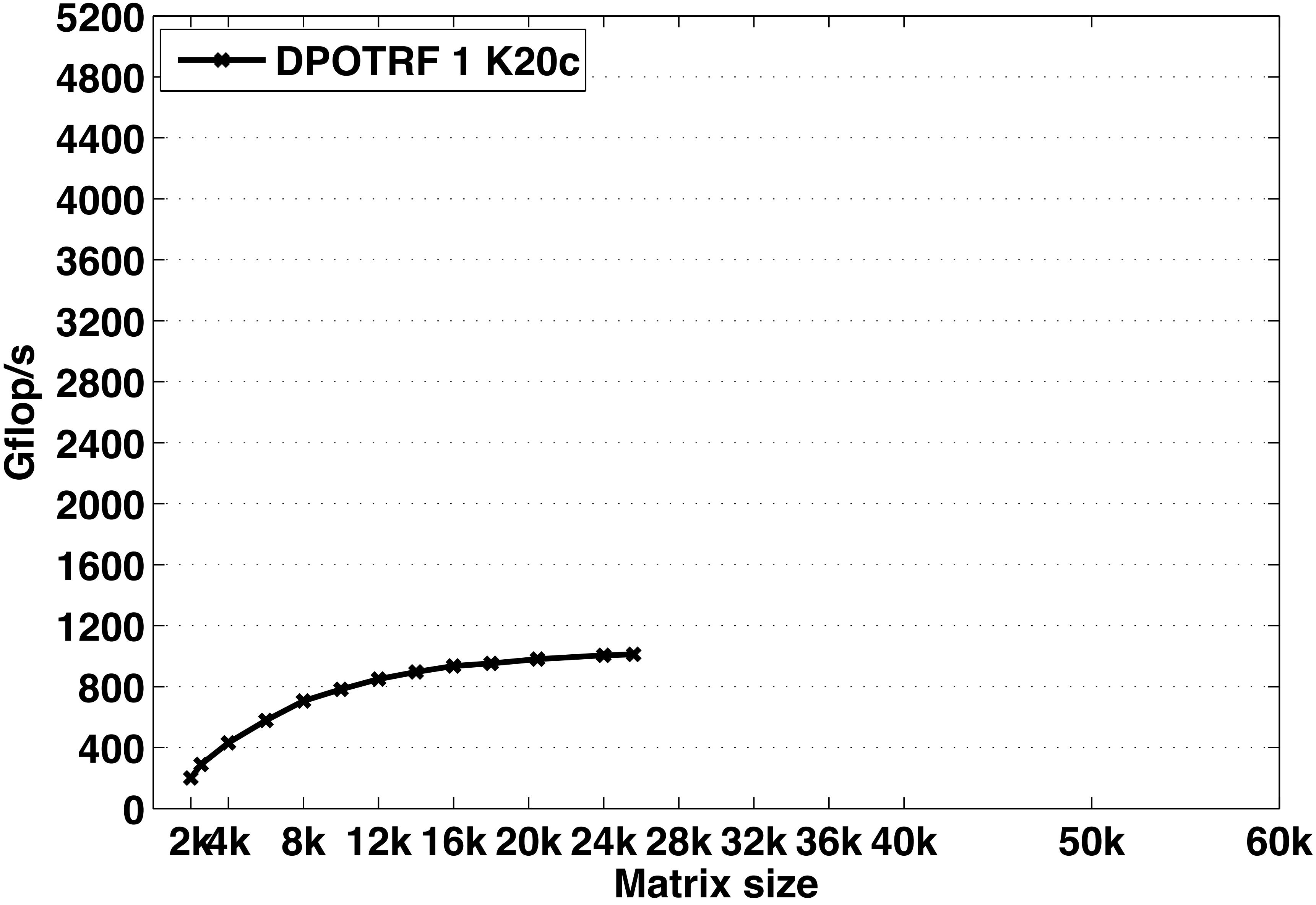
➢snapshot of the execution trace of the Cholesky factorization on System A for a matrix of size 40K using six GPUs K20c.

➢As expected the pattern of the trace looks compressed which means that our implementation is able to schedule and balance the tasks on the whole six GPUs devices.

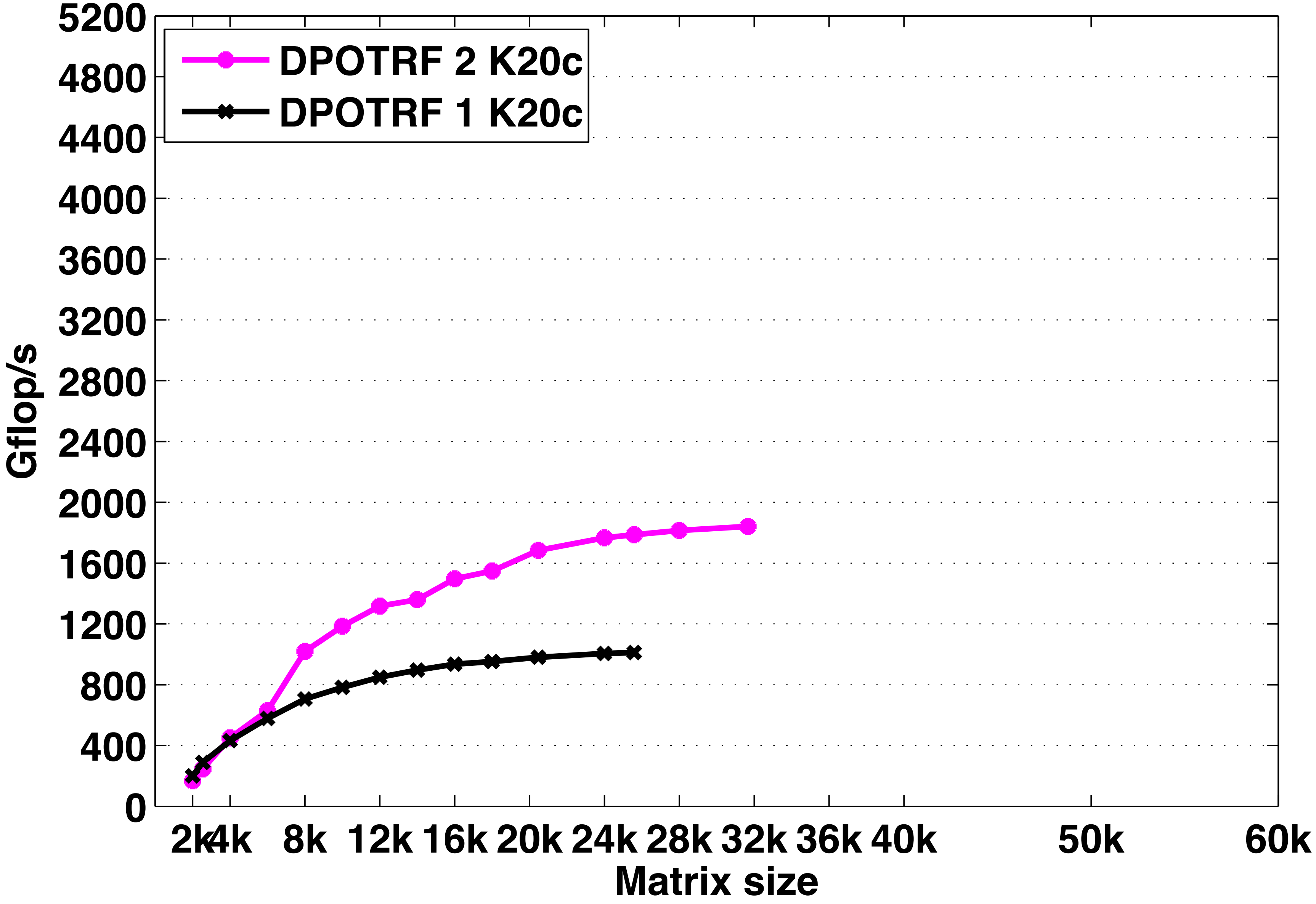
High Performance Computing : current development

The Cholesky factorization DPOTRF

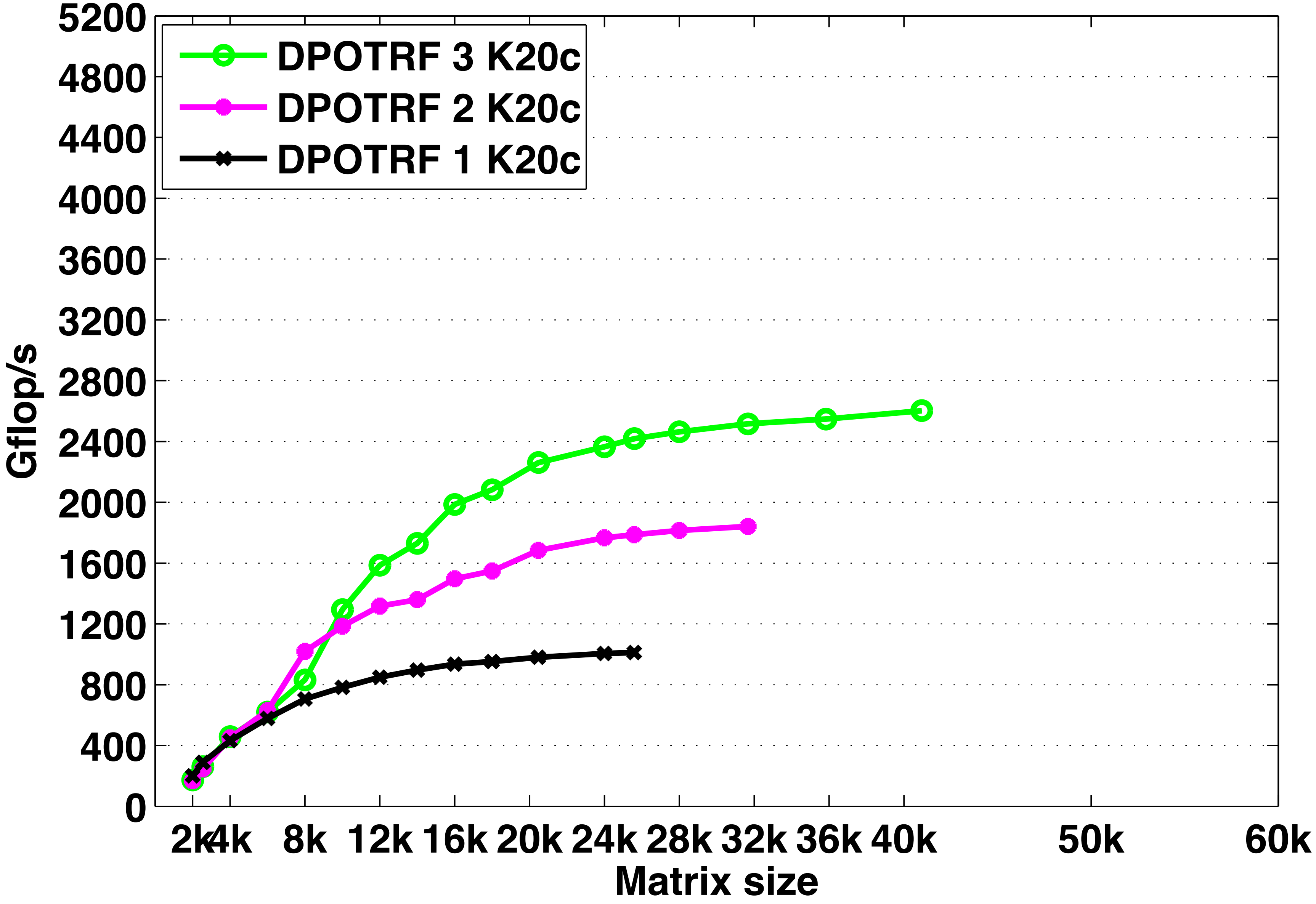
magma_quark scalability DPOTRF Kepler K20c



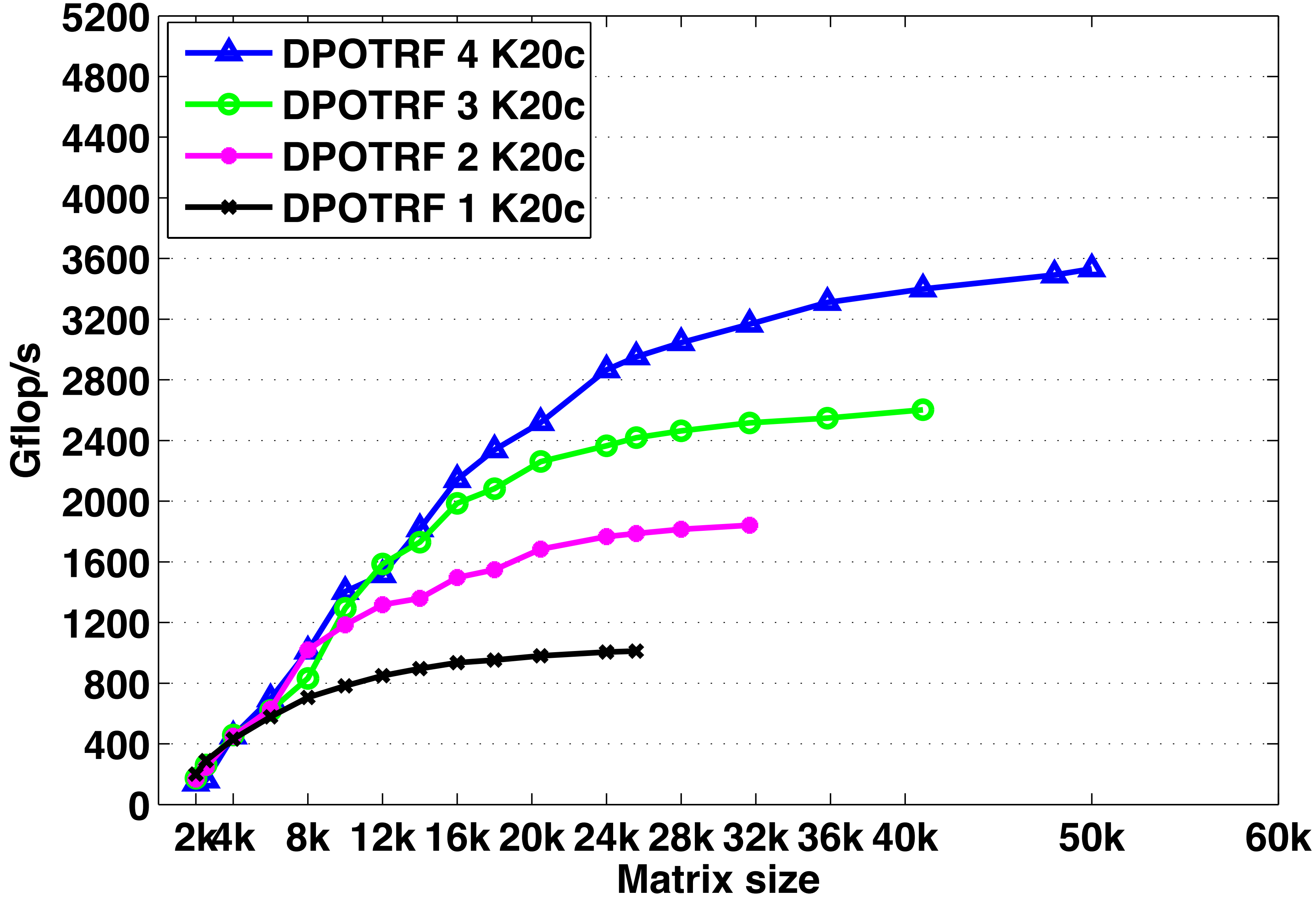
magma_quark scalability DPOTRF Kepler K20c



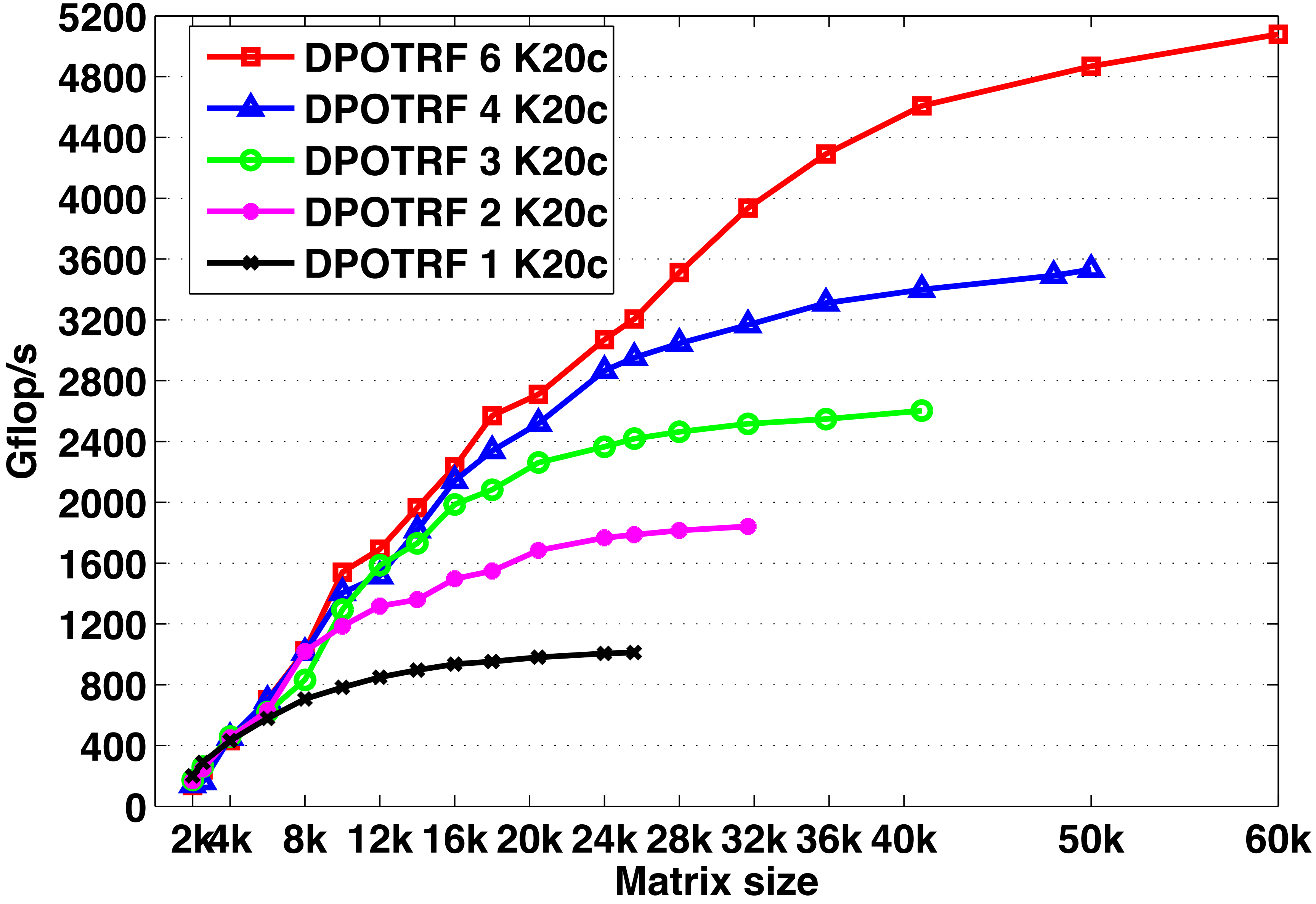
magma_quark scalability DPOTRF Kepler K20c



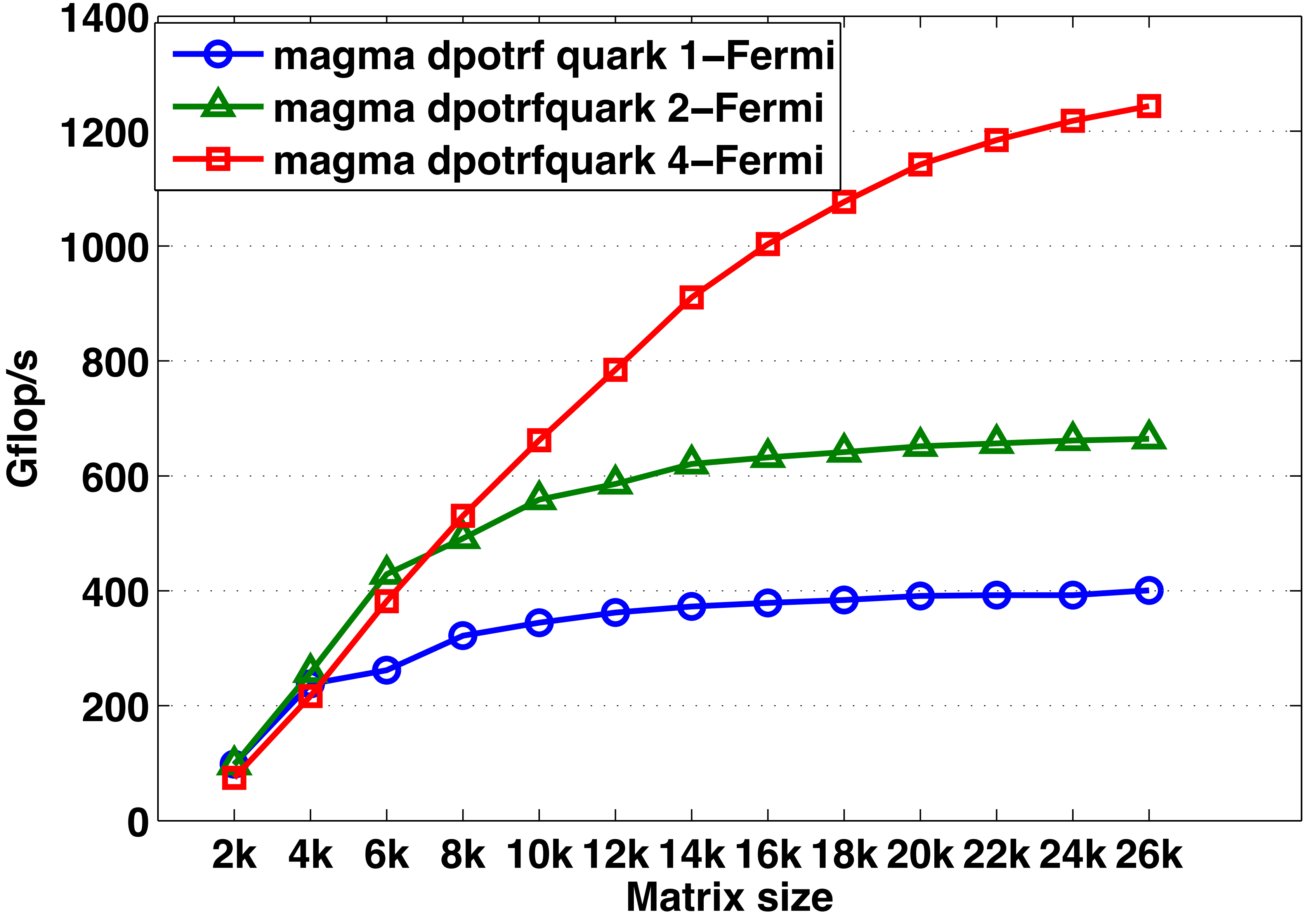
magma_quark scalability DPOTRF Kepler K20c



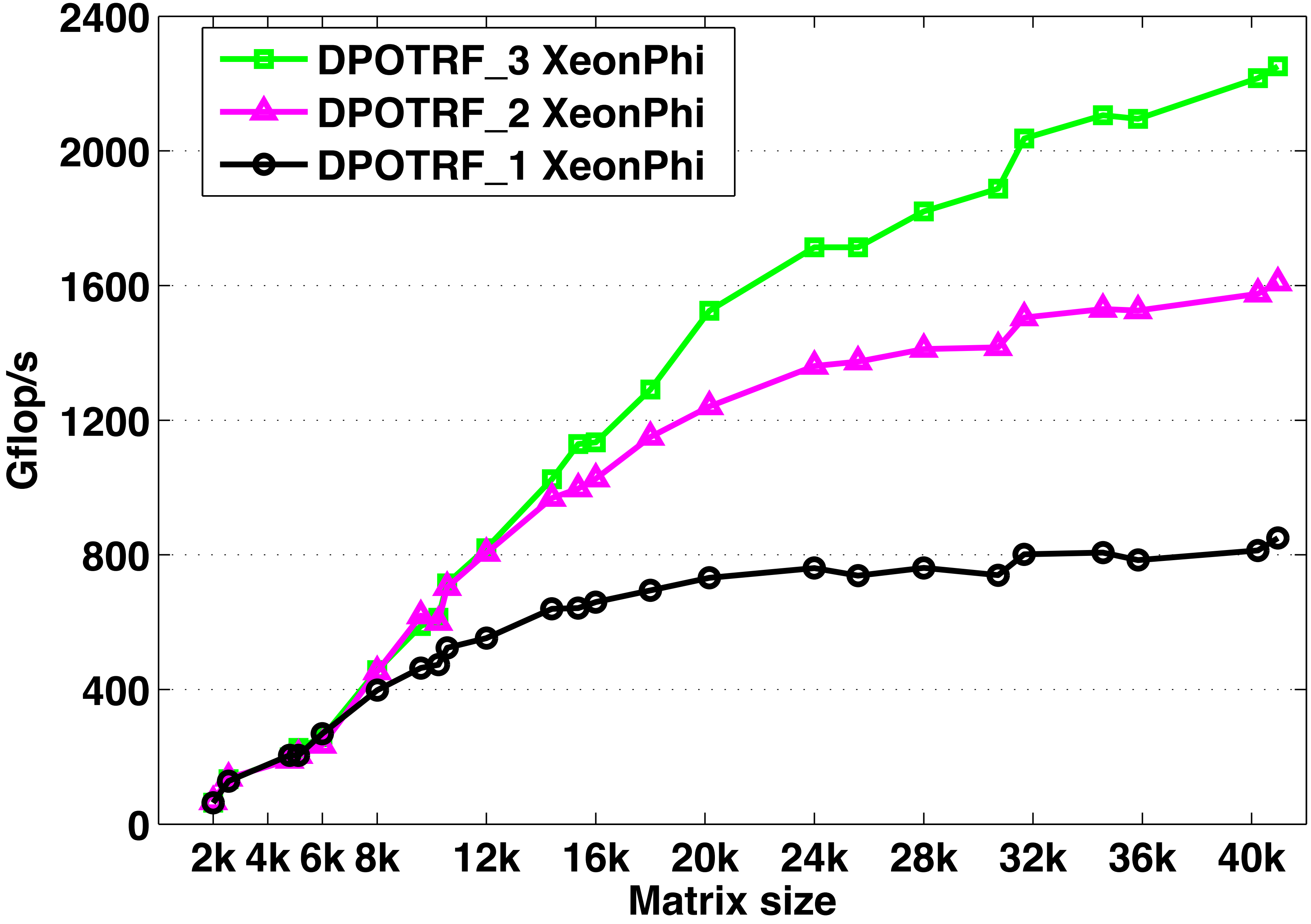
magma_quark scalability DPOTRF Kepler K20c



magma_quark scalability DPOTRF Fermi M2090



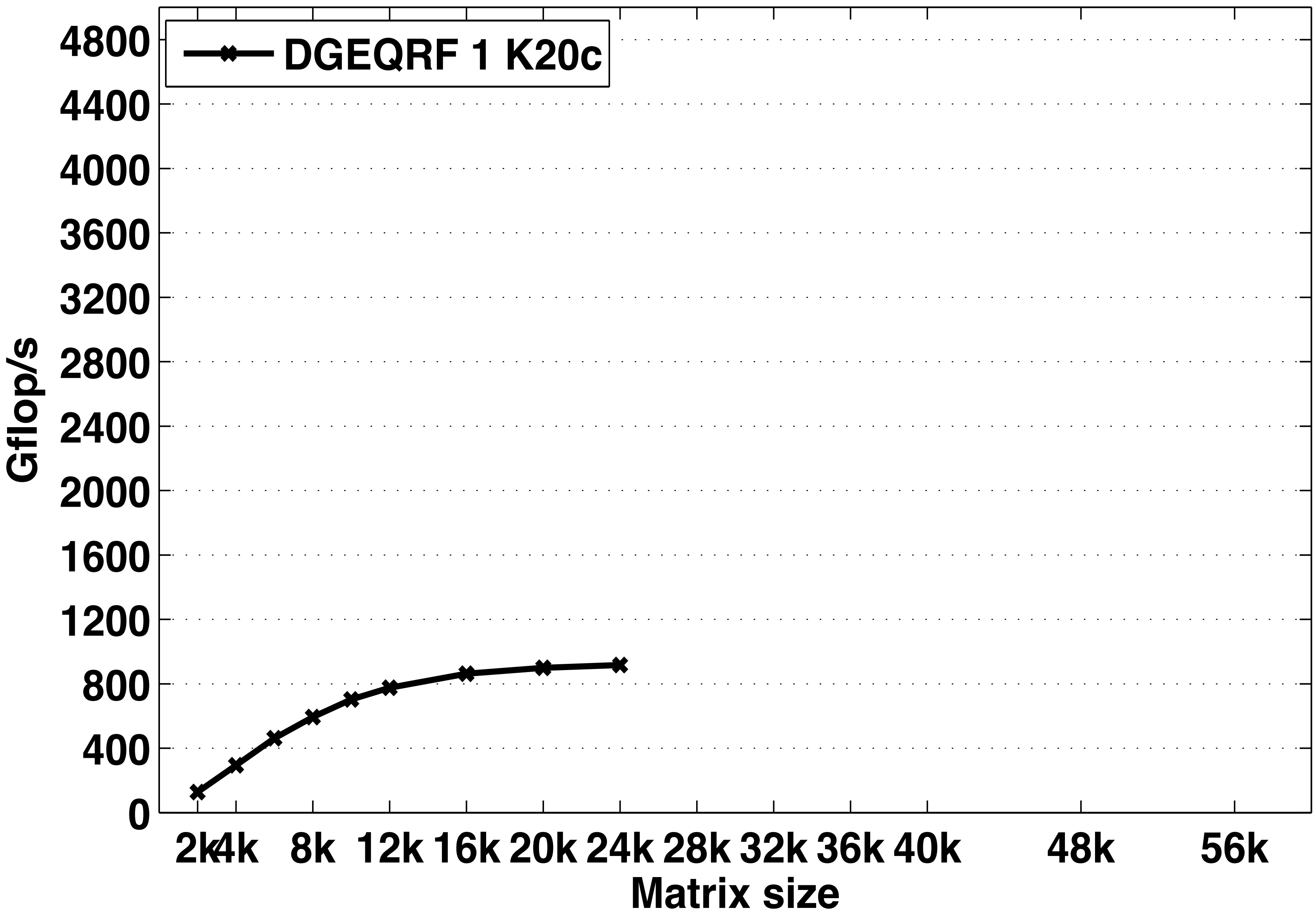
magma_quark scalability DPOTRF Xeon-Phi



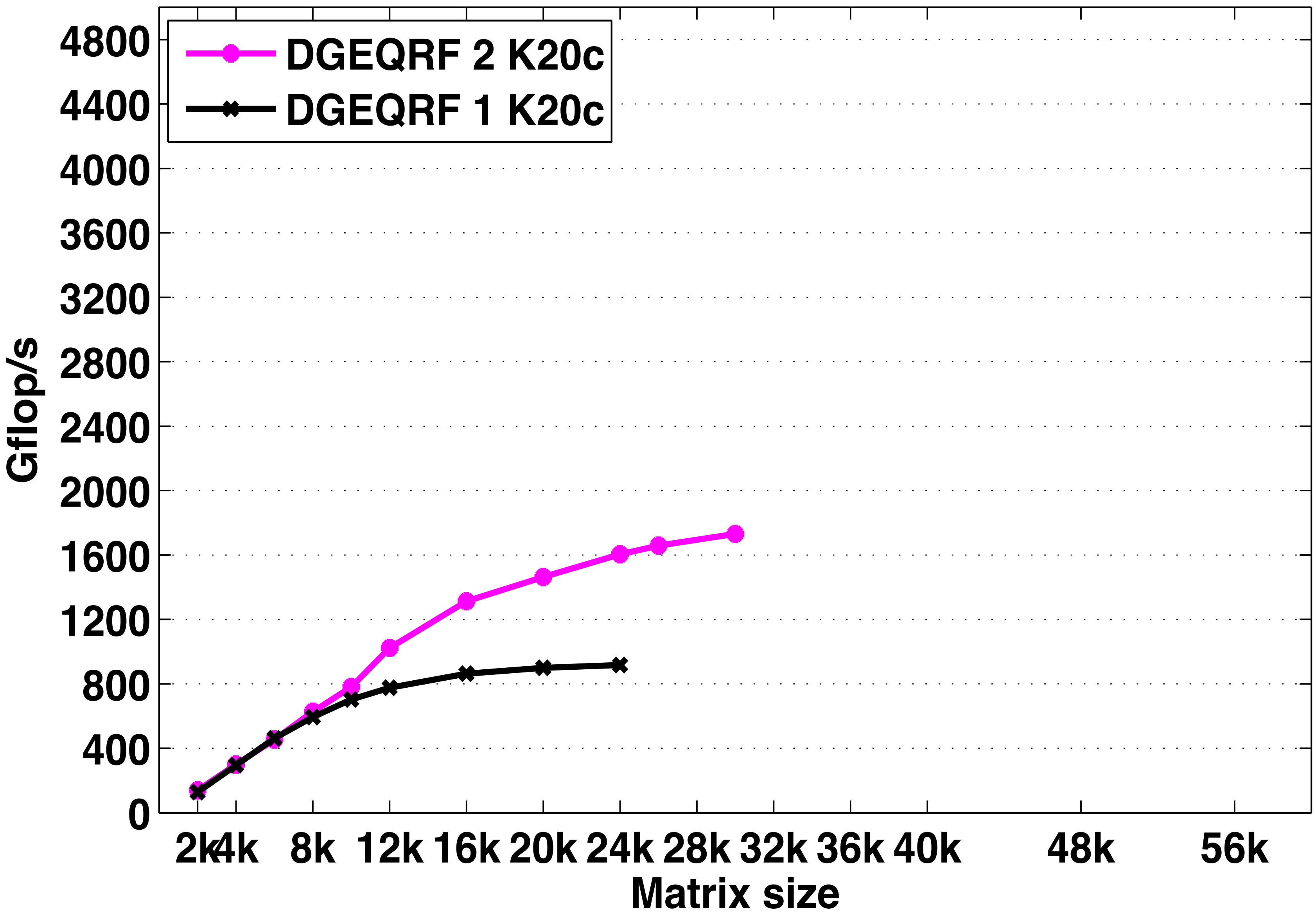
High Performance Computing : current development

The QR decomposition DGEQRF

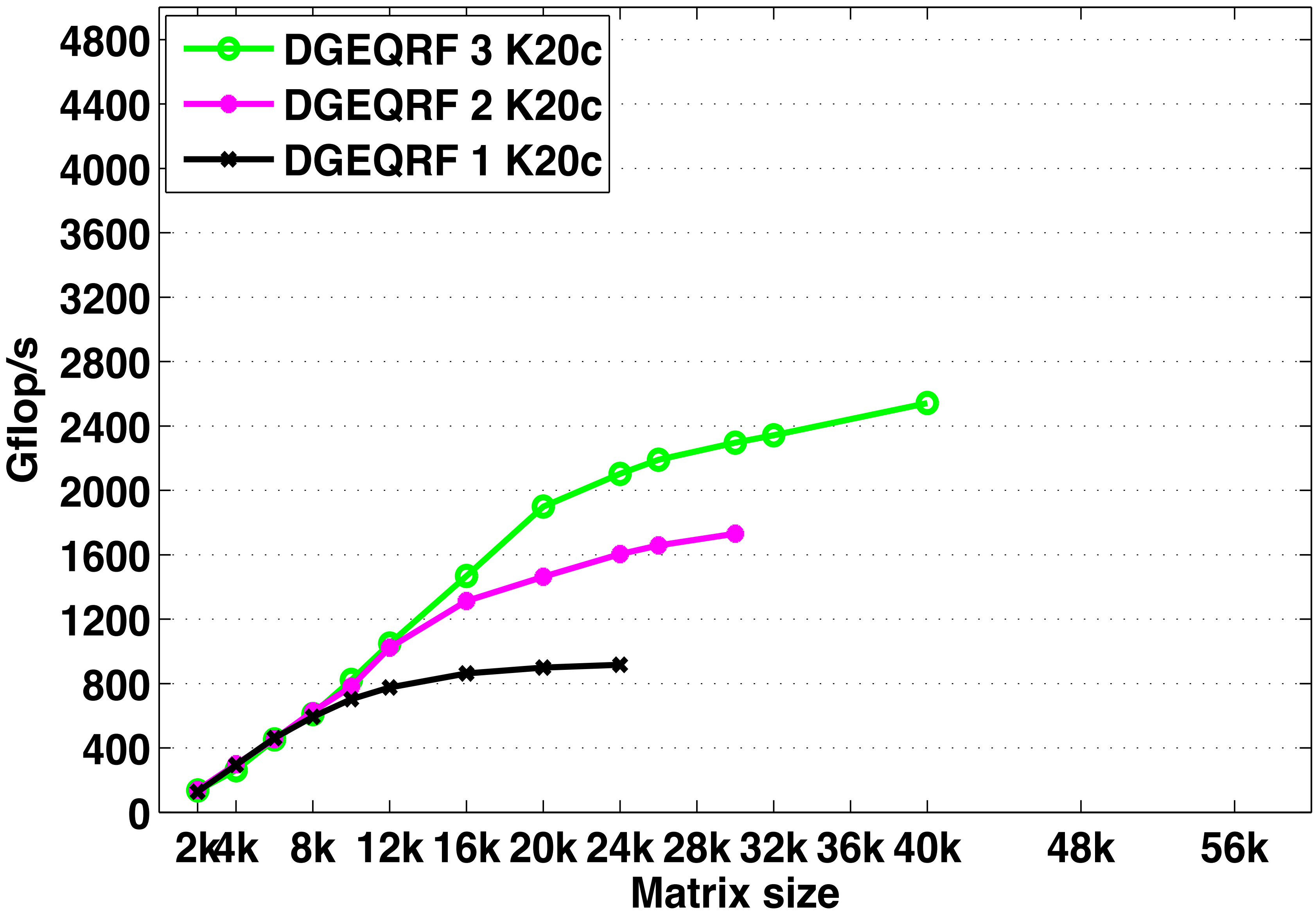
magma_quark scalability DGEQRF Kepler K20c



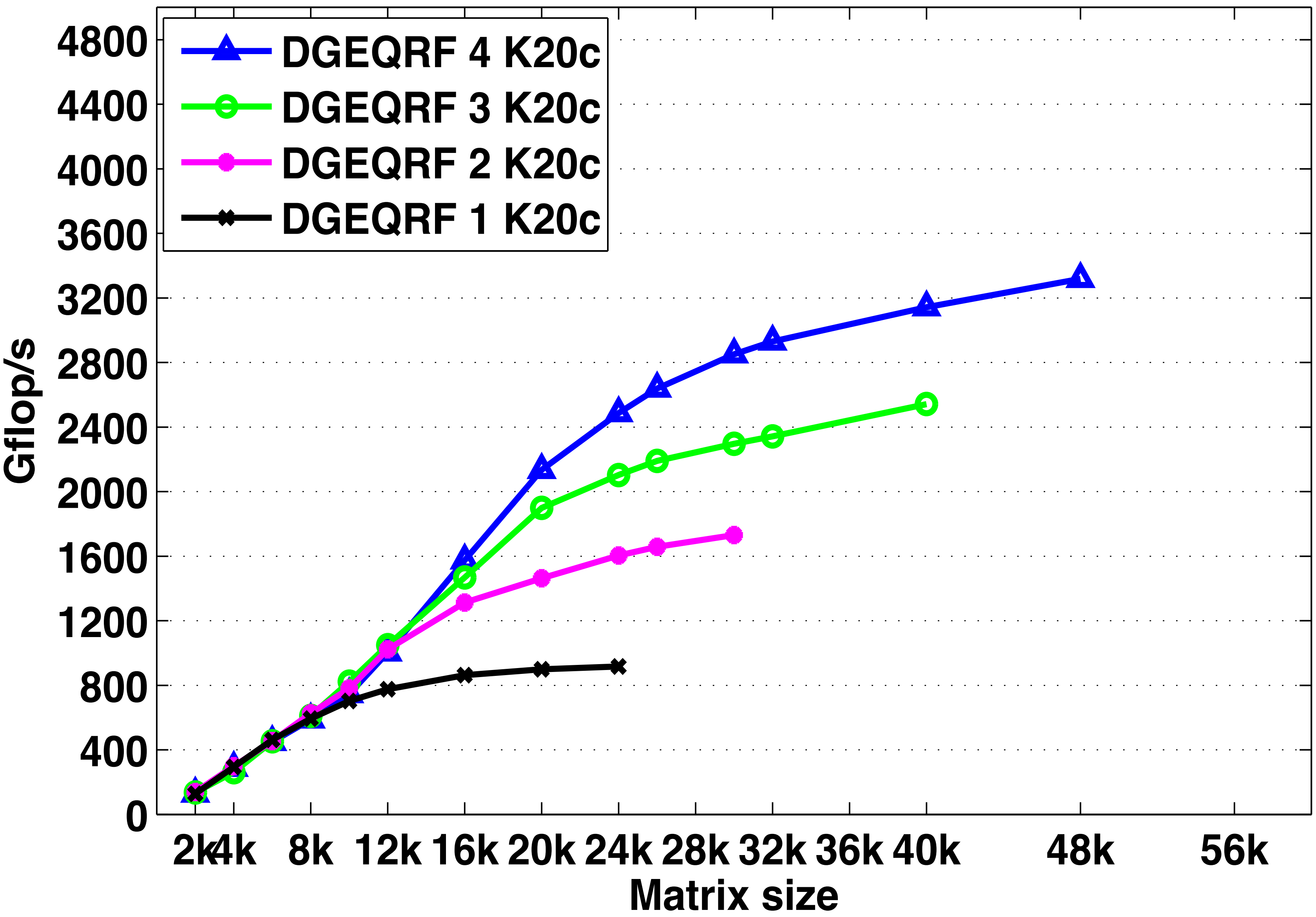
magma_quark scalability DGEQRF Kepler K20c



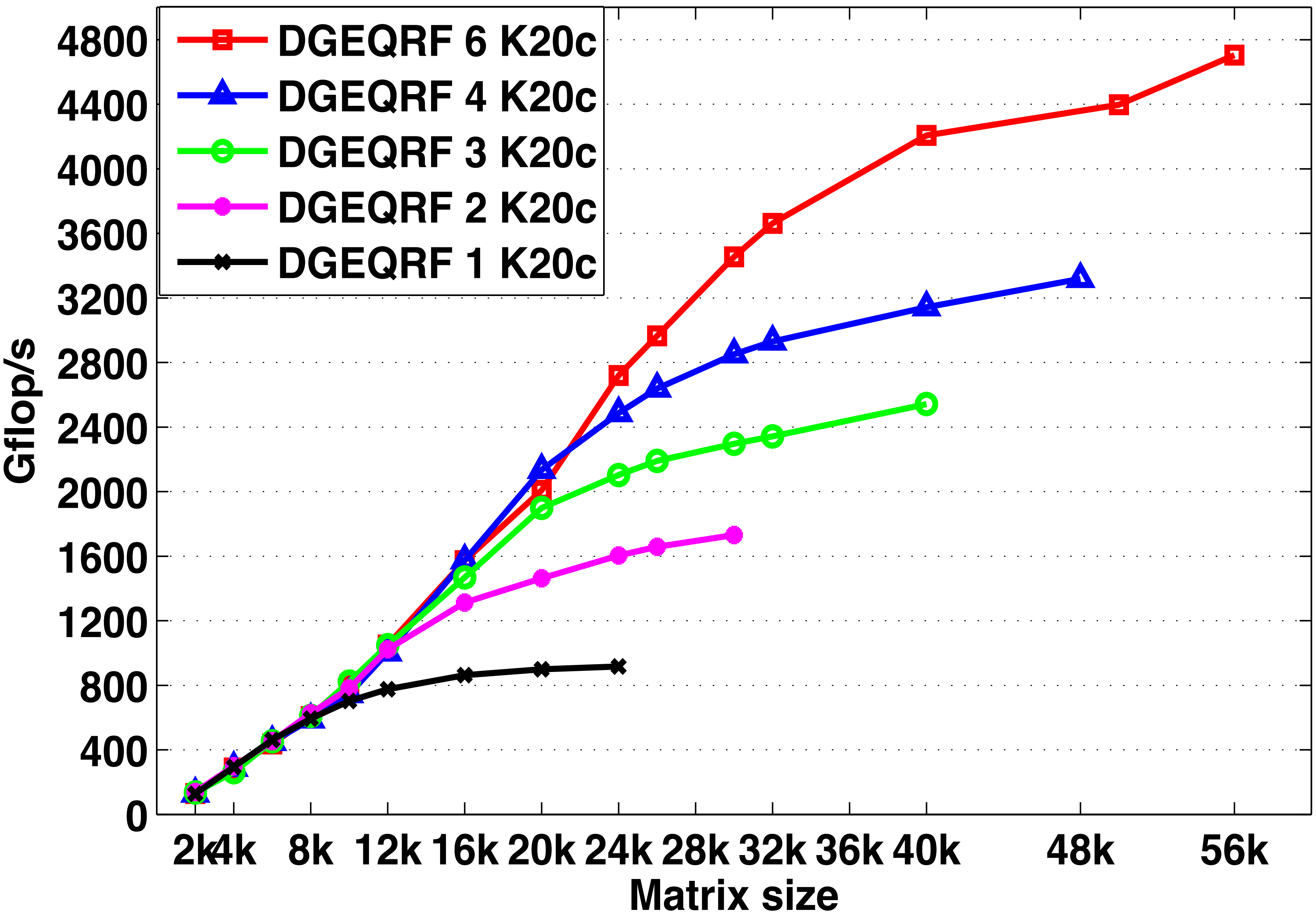
magma_quark scalability DGEQRF Kepler K20c



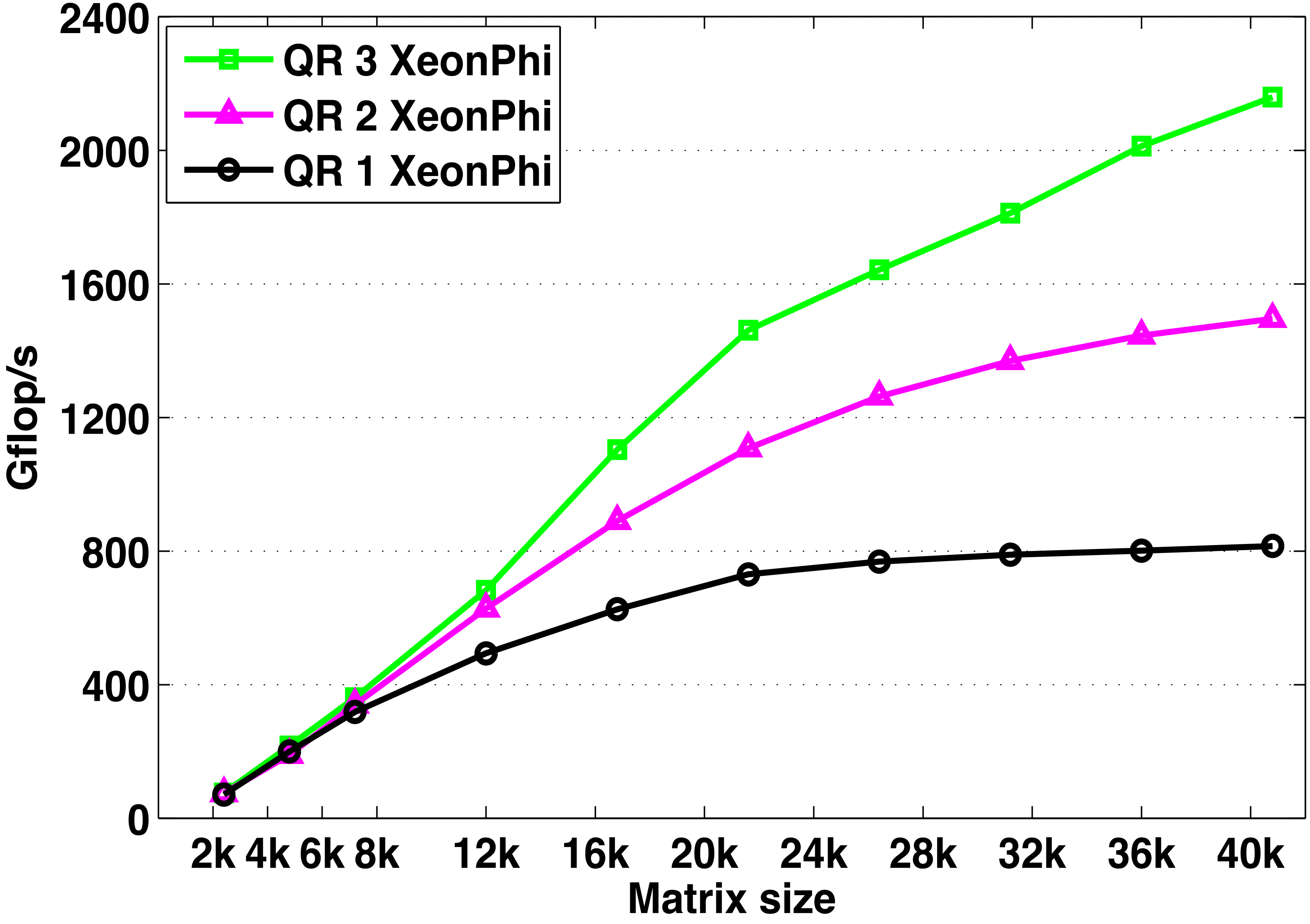
magma_quark scalability DGEQRF Kepler K20c



magma_quark scalability DGEQRF Kepler K20c



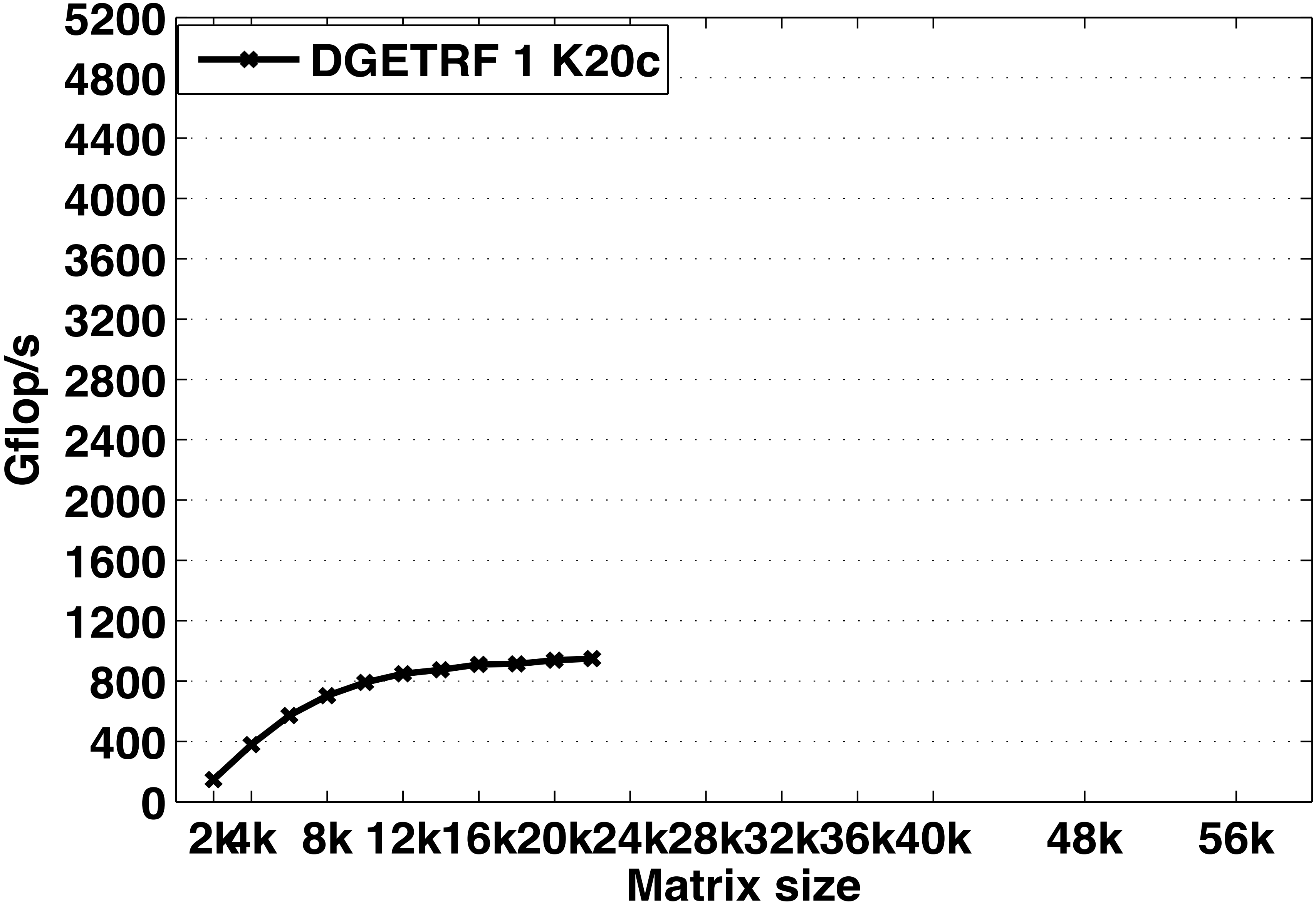
magma_quark scalability DGEQRF Xeon-Phi



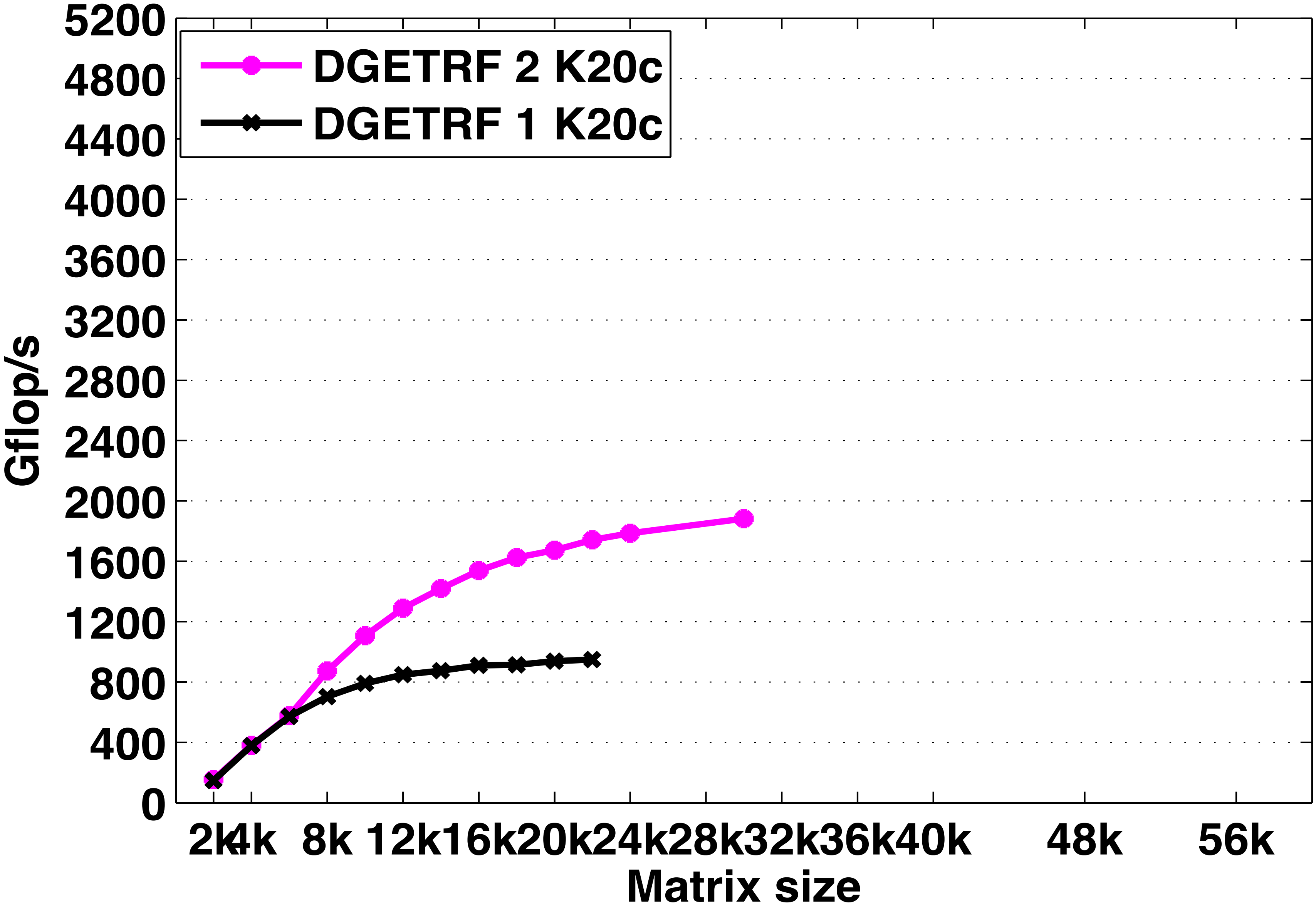
High Performance Computing : current development

The LU factorization DGETRF

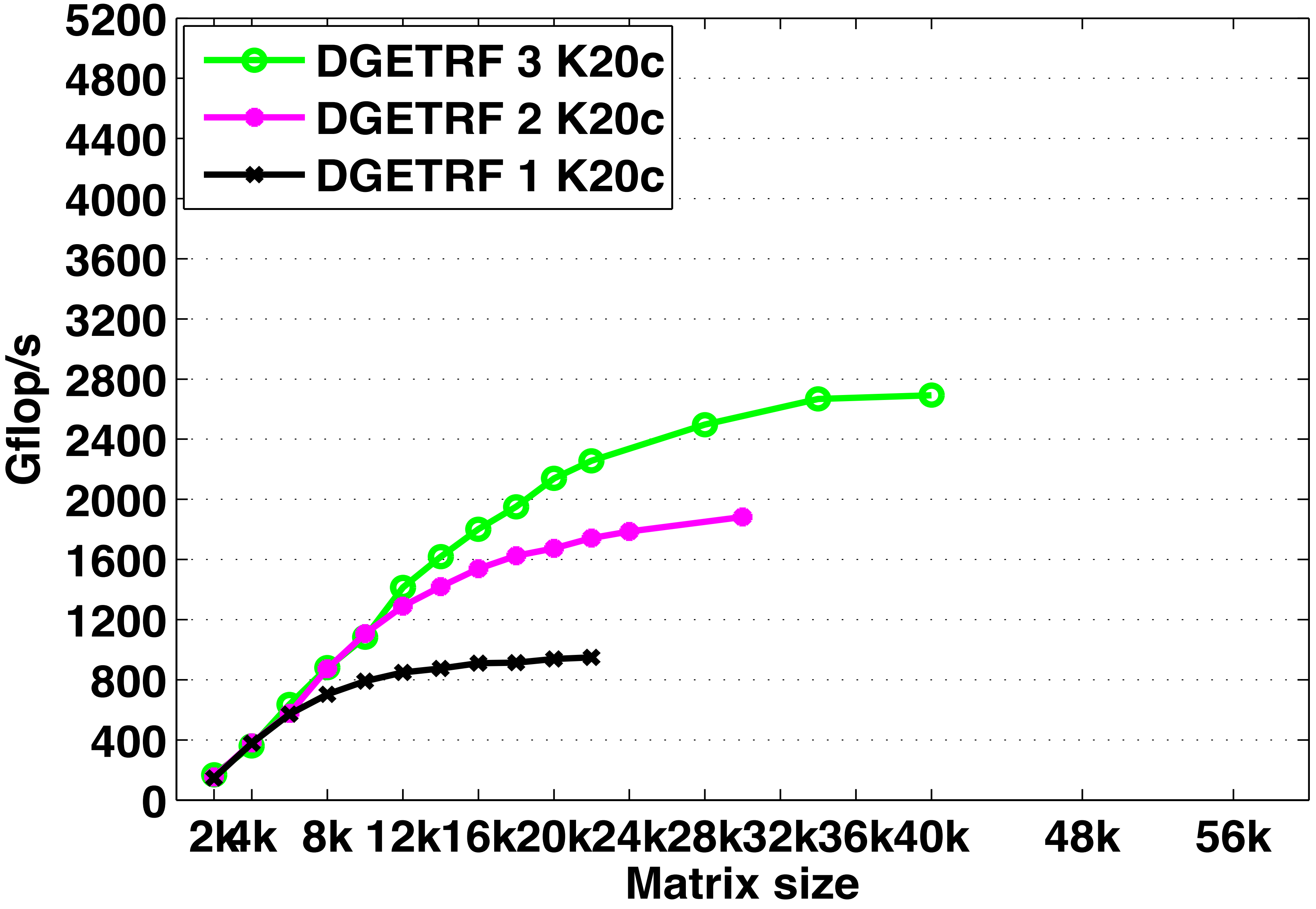
magma_quark scalability DGETRF Kepler K20c



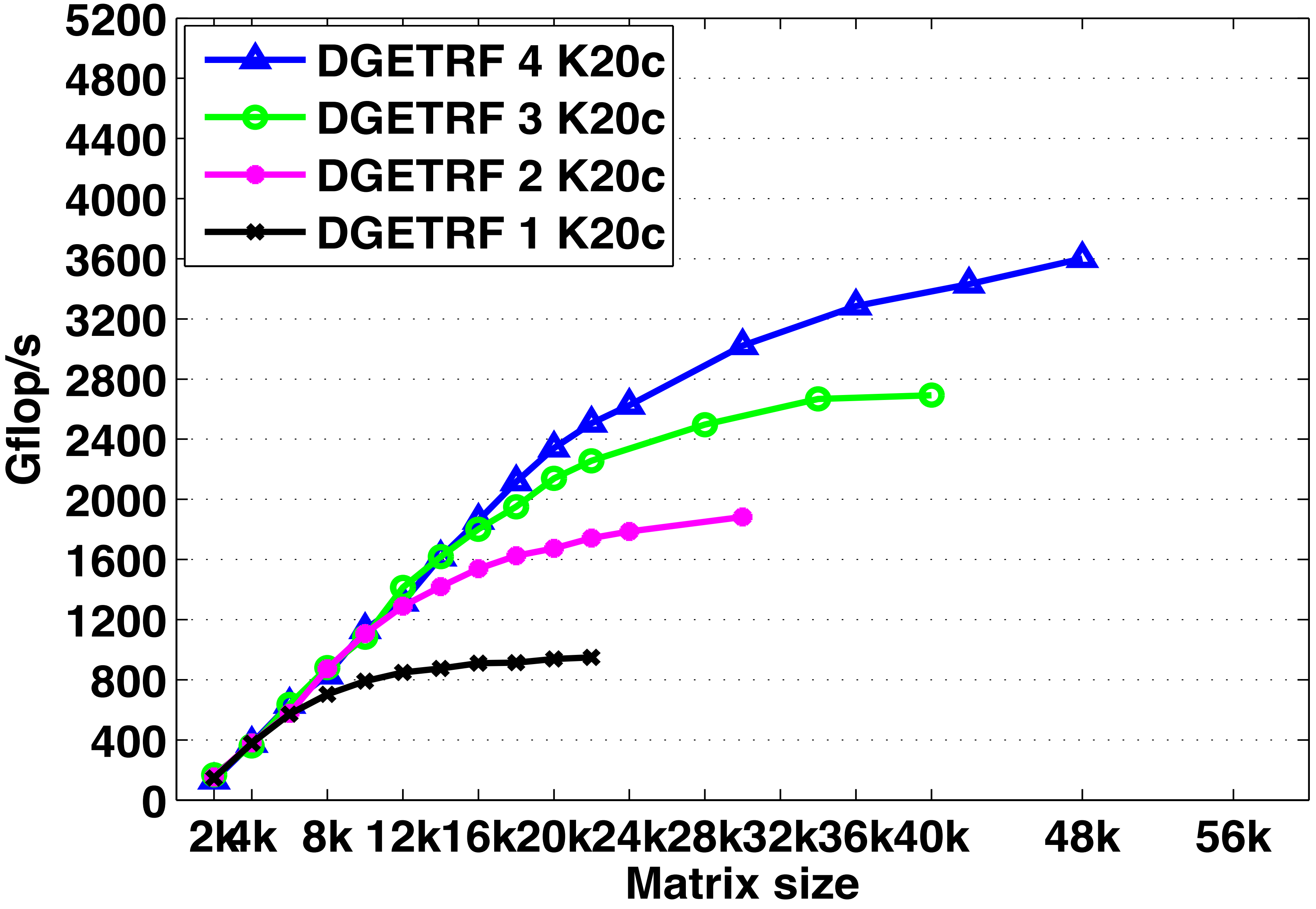
magma_quark scalability DGETRF Kepler K20c



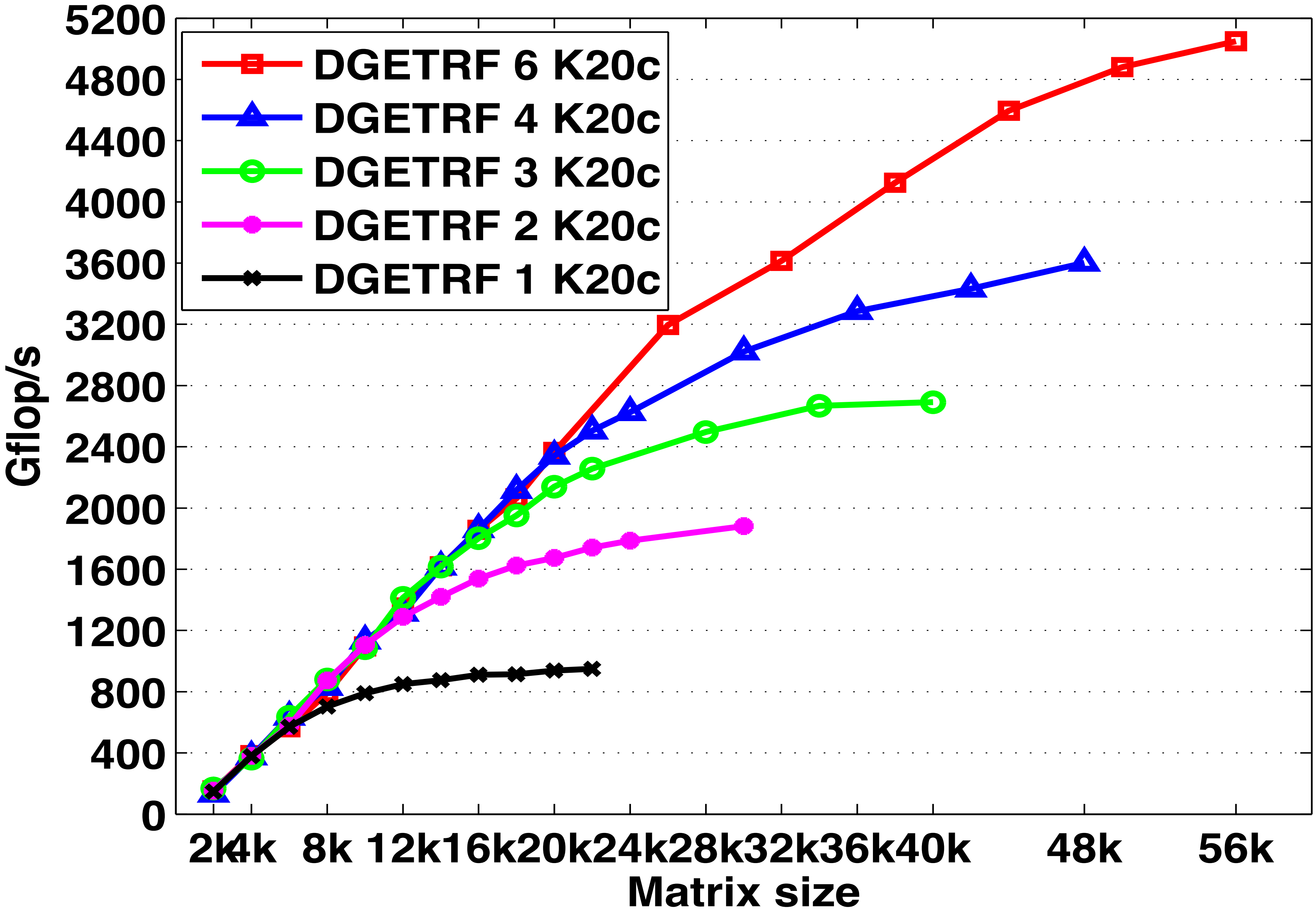
magma_quark scalability DGETRF Kepler K20c



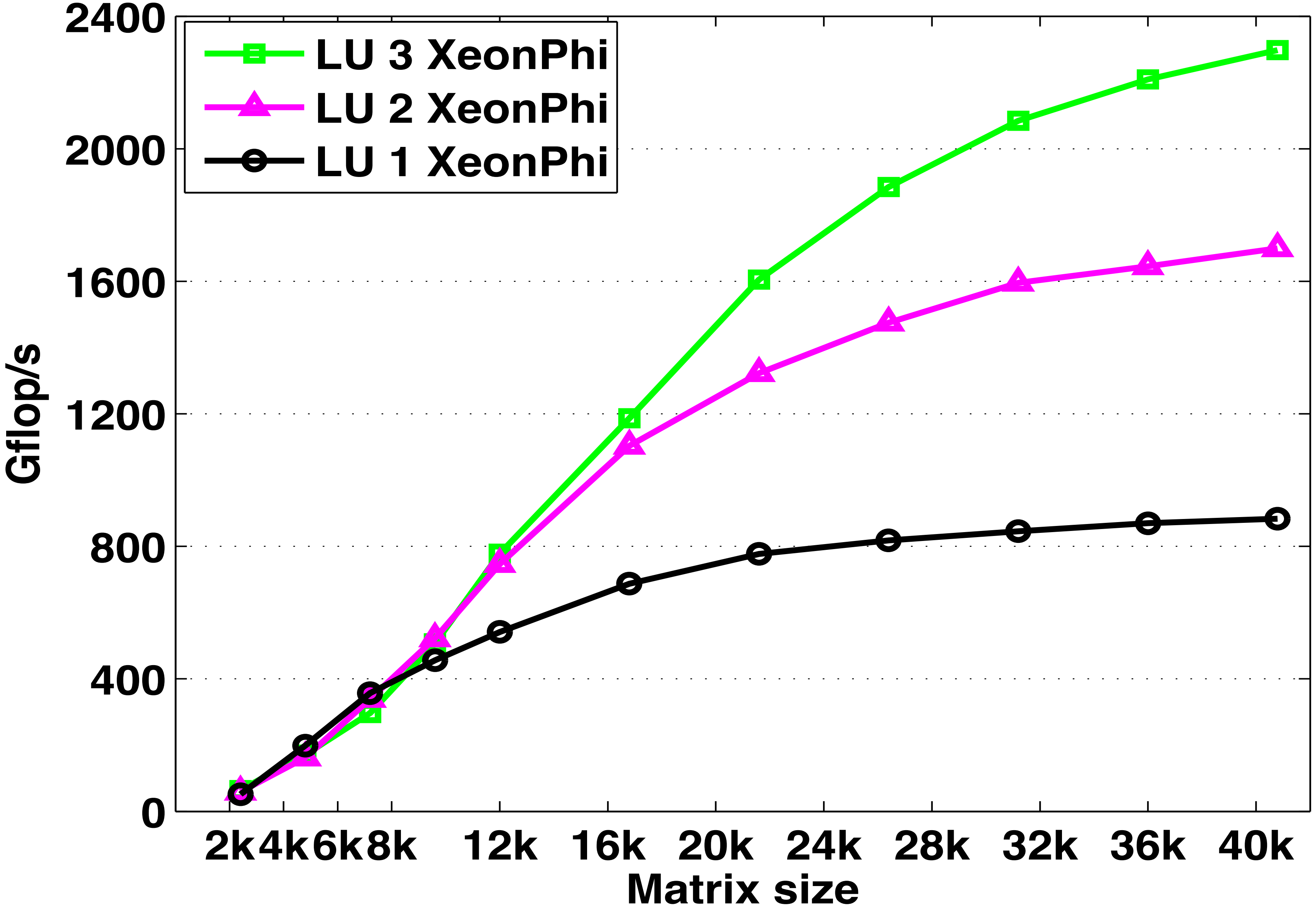
magma_quark scalability DGETRF Kepler K20c



magma_quark scalability DGETRF Kepler K20c



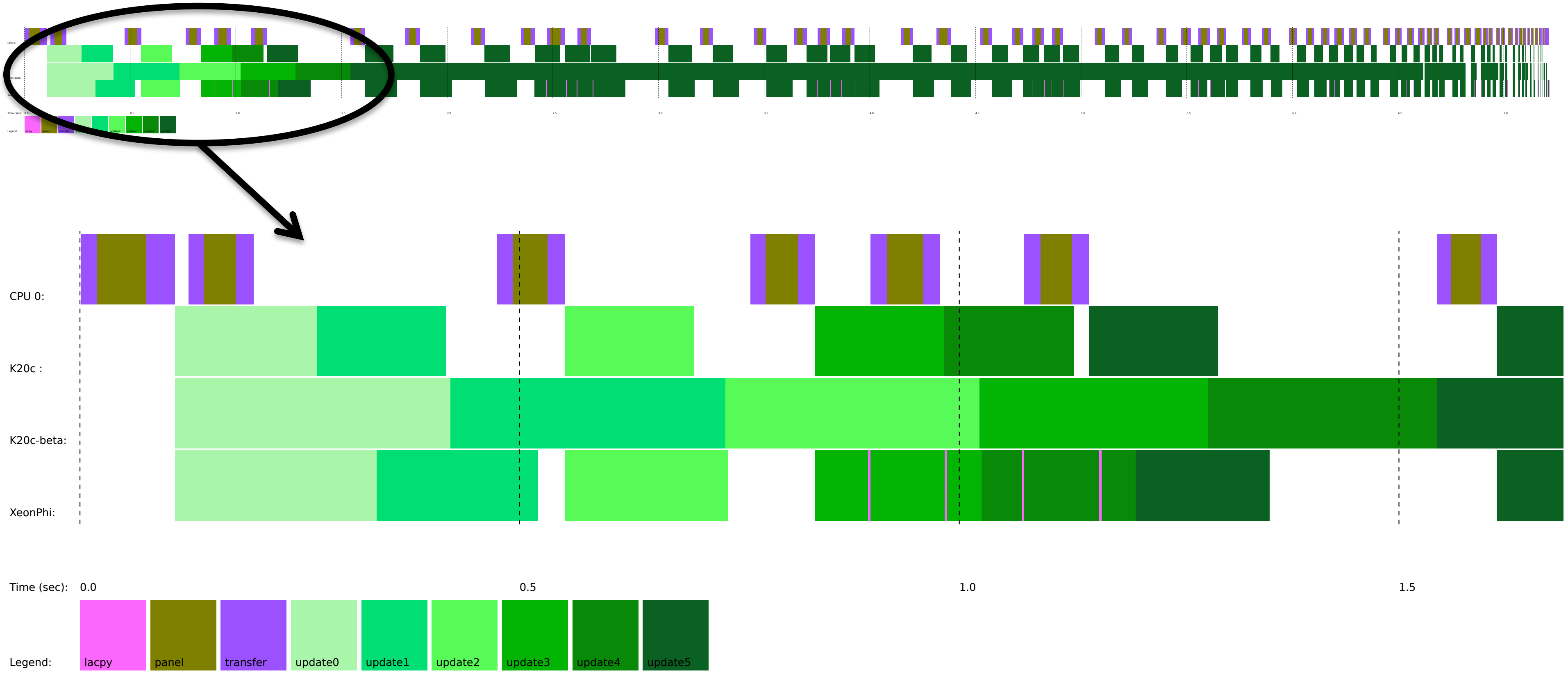
magma_quark scalability DGETRF Xeon-Phi



High Performance Computing : current development

What about
multi-way heterogeneous

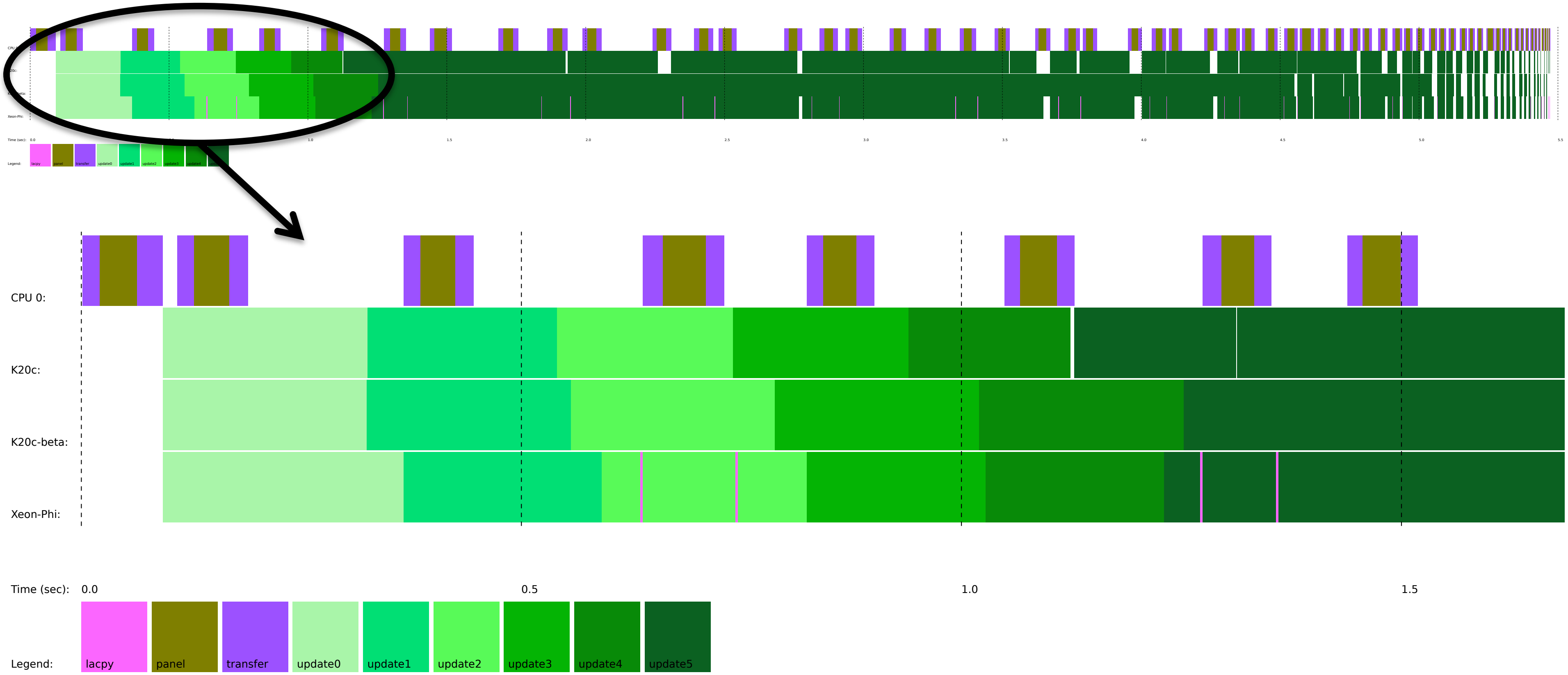
High Performance Computing : current development



Hardware guided data distribution and load balance :

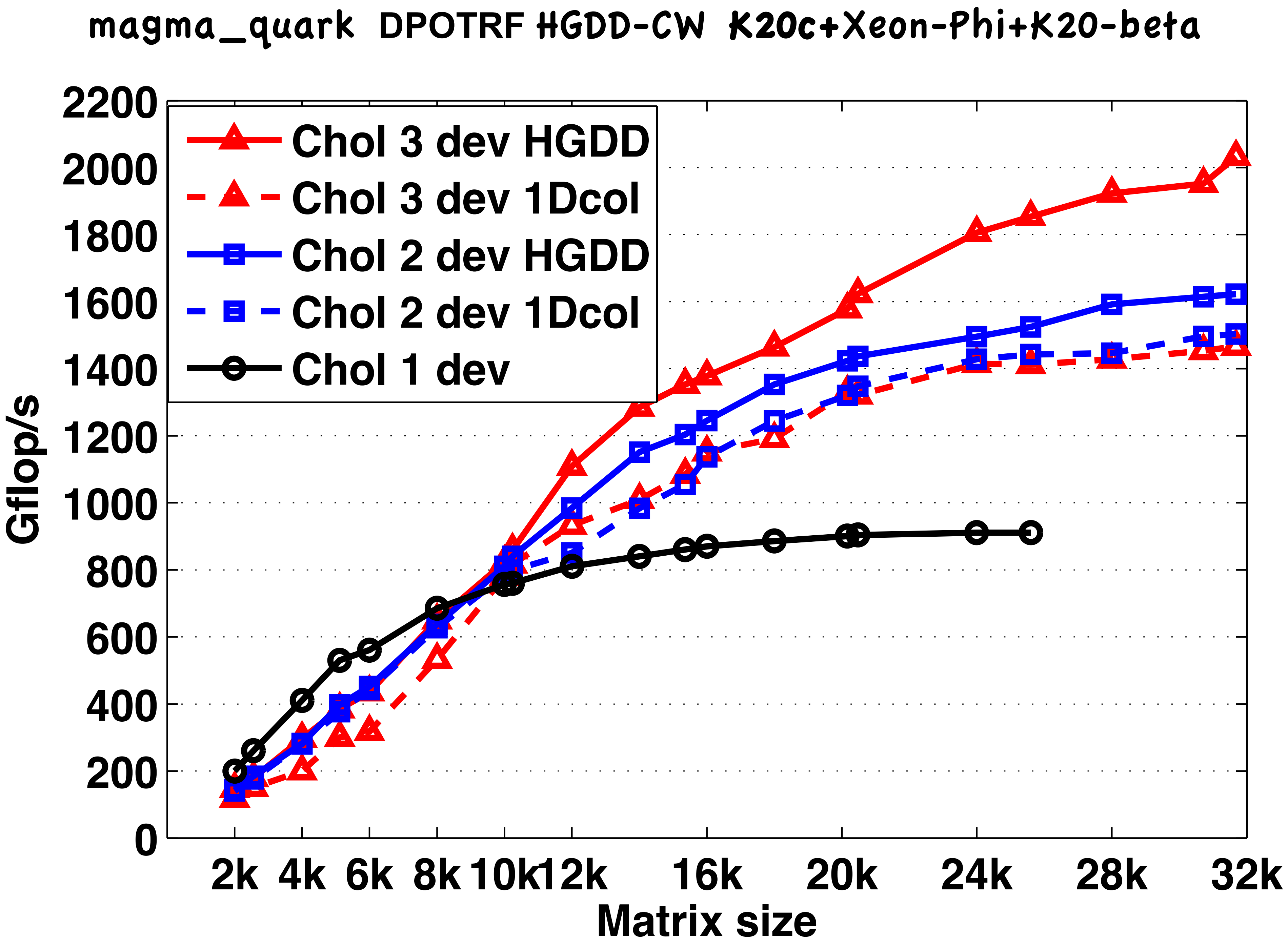
➢Using the QUARK runtime, the data is either distributed or redistributed in an automatic fashion so that each device gets the appropriate volume of data to match its capabilities.

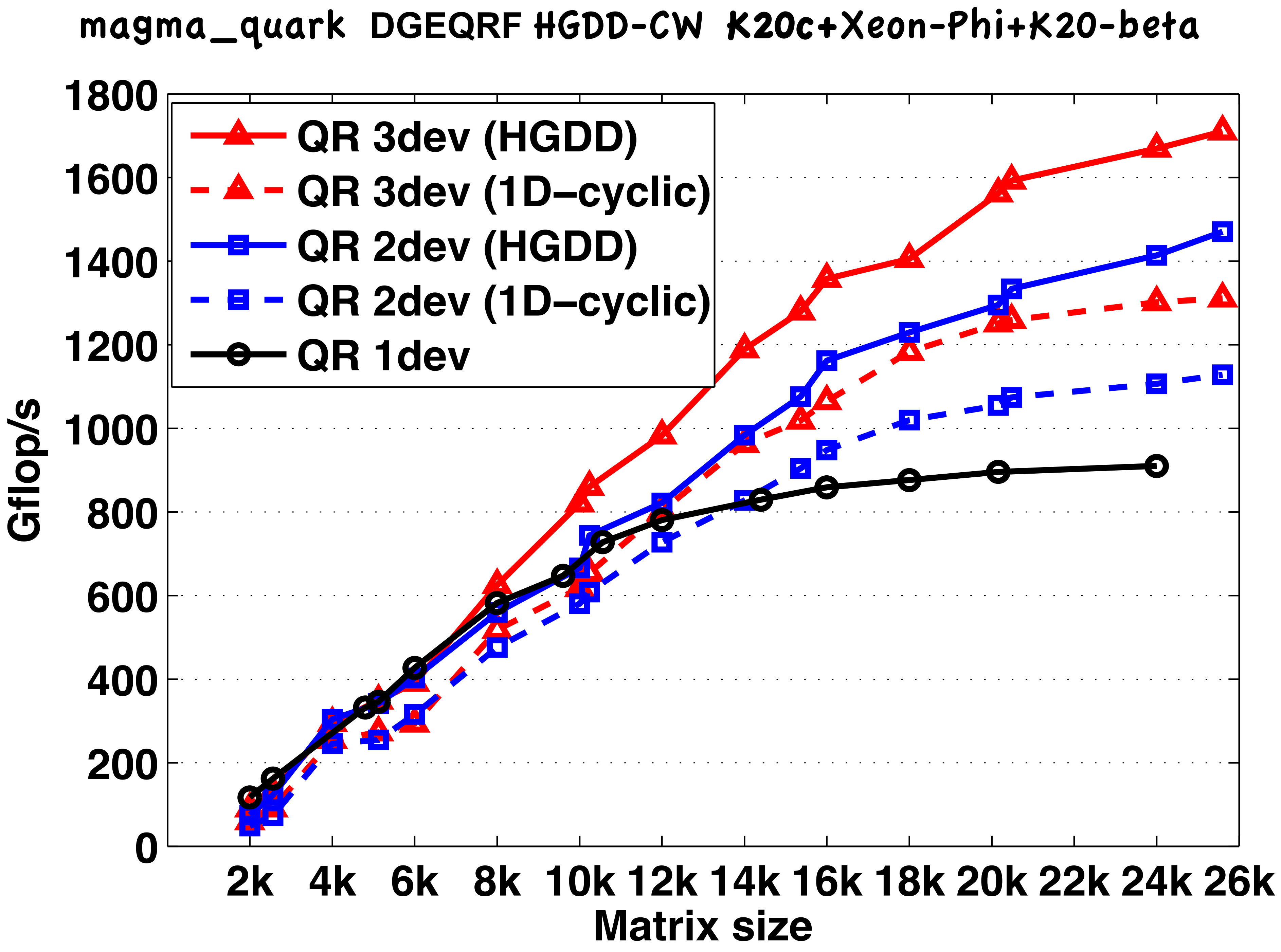
High Performance Computing : current development



Hardware guided data distribution and load balance :

➢Using the QUARK runtime, the data is either distributed or redistributed in an automatic fashion so that each device gets the appropriate volume of data to match its capabilities.


magma_quark DPOTRF HGDD-CW K20c+Xeon-Phi+K20-beta
Chol 3 dev HGDD
Chol 3 dev 1Dcol
Chol 2 dev HGDD
Chol 2 dev 1Dcol
Chol 1 dev
Gflop/s
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
2k
4k
6k
8k
10k
12k
16k
20k
24k
28k
32k
Matrix size


magma_quark DGEQRF HGDD-CW K20c+Xeon-Phi+K20-beta
QR 3dev (HGDD)
QR 3dev (1D–cyclic)
QR 2dev (HGDD)
QR 2dev (1D–cyclic)
QR 1dev
Gflop/s
0
200
400
600
800
1000
1200
1400
1600
1800
2k
4k
6k
8k
10k
12k
14k
16k
18k
20k
22k
24k
26k
Matrix size

High Performance Computing : current development

Questions ?