# Tiled QR Decomposition and Its Optimization on CPU and GPU Computing System

Dongjin Kim and Kyu-Ho Park
Computer Engineering Research Lab., Electrical Engineering
Korea Advanced Institute of Science and Technology
Daejeon, Republic of Korea
djkim@core.kaist.ac.kr and kpark@ee.kaist.ac.kr

***Abstract*—There can be many types of heterogeneous computing systems, and the most useful one is the CPU and GPU computing system. In this system, we try to run QR decomposition, which expresses a standard real matrix as a production of two matrices. For a tiled QR decomposition algorithm, which is a parallelized version of QR decomposition, because of the heterogeneity of computing devices and communication cost, the way that each tile is distributed into which device is the main issue of tiled QR decomposition. The goal of this study is to optimize the tile distribution and the tiled QR decomposition operation mathematically, depending on the given system. We select the main computing device for the main steps of the algorithm, optimize the number of devices, and optimize the tile distribution among the devices using a distribution guide array. Our evaluation confirms that our method has good scalability and the optimization process maximizes the tiled QR decomposition performance.**

## I. Introduction

As time goes on, the performance and parallelism of computing devices, such as CPU, GPU, Xeon Phi [1], or other types of devices, are improving faster and faster. According to this trend, it is very good to use heterogeneous devices together to solve a single problem that needs fast computing speed and a highly parallel computing environment.

There can be many types of heterogeneous computing systems, and our work focuses on a CPU and GPU (or GPGPU) heterogeneous computing system. Fig. 1 shows the simplified architecture of the system, with the GPU based on CUDA [2], [3] for our target system, which is a parallel computing architecture of NVIDIA GPUs. There are one or several cores within a single CPU, and all the CPUs can easily share data in their memory, including the main memory and disk. In GPUs, there are hundreds or thousands of cores within a single GPU, and each GPU can access data only in its own local memory. Of course, GPU cores within a single GPU can share their data using the local memory.

In terms of computing environment, the operation of each GPU is subordinate to the operation of a CPU. To run part of a program of a GPU, also known as the kernel, an invocation from the CPU is necessary. The CPU first transfers needed data to the GPU and then invokes the GPU's operation; the result will be transferred from the GPU to the CPU. Additionally, the operation on the GPU is not preemptive. To run multiple kernels on a single GPU simultaneously, other kernels must wait until the current kernel finishes its current processing on the GPU.

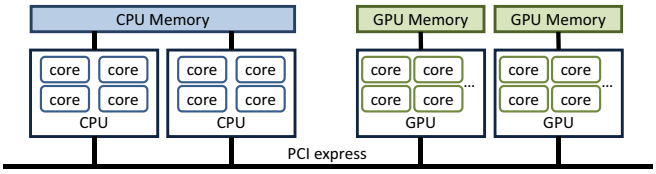



Fig. 1. CPU and GPU heterogeneous computing system architecture

This system has two major properties that have to be considered for better operation. The first one is the different computation environment of each computing device. Because of different parameters, such as the core architecture, clock speed or memory bandwidth, some jobs can be calculated faster on the CPU and others on the GPU. Numbers of parallel cores of GPU need jobs to be able to be computed in parallel, and CPU cores are good for some jobs with low parallelism or more memory dependent jobs. The second major property is the need of a memory copy. Since the CPU cannot access the GPU's memory directly and vice versa, there must be a memory copy to share some data between the CPU and GPU or among GPUs through PCI express communication. If there are too many data to share during processing, the communication cost can be a bottleneck, and the utilization of both the CPU and GPU might be low.

Our target application is a QR decomposition operation. QR decomposition, also known as QR factorization, expresses a standard real matrix as a production of two matrices, $Q$ and $R$, where $Q$ is an orthogonal matrix and $R$ is an upper triangular matrix. This decomposition operation is the basis for solving some systems of linear equations, so it is widely used in data analysis of various domains.

There are several types of QR decomposition, such as the Householder or Cholesky methods. Our work mainly relies on the Householder reflections method [4], because it is efficient and well-matching with parallel computations. From the left side to the right side, reflectors are computed, and each element on the under-diagonal part of each column vector is reduced by the Householder matrix.

To parallelize the QR decomposition, using a tiled QR

0190-3918/13 $26.00 © 2013 IEEE
DOI 10.1109/ICPP.2013.88

decomposition algorithm [5], [6] is usual. In the algorithm, a given matrix is divided into several tiles, and groups of tiles are distributed into each computing device to operate parallelized QR decomposition. During the distribution, because of the heterogeneity of computing devices and communication cost, the way that each tile is distributed into which device is the main issue of tiled QR decomposition.

The goal of this study is to optimize tile distribution and the tiled QR decomposition operation mathematically, depending on the given system. The main contributions of our work are as follows.

- We divide the QR decomposition of a single tile into several steps, depending on the processing properties and operate each step on an appropriate computing device. During this process, we select the main computing device, which operates the main steps of the decomposition.
- We optimize the number of devices that participate in the tiled QR decomposition. Depending on the input matrix, some of the available computing devices may not be used. For optimization, the processing speed of each device and the communication cost among the devices are considered to calculate the time tradeoff.
- For tile distribution, we construct a distribution guide array, which contains the indications of "which tiles are on which device". To build this array, the processing speed and parallelism of each device are considered.

The remaining part of this paper is organized as following. In Section II, we explain more detail of QR decomposition and tiled QR decomposition. In Section III, we introduce our motivations. Our contributions are explained mainly in Section IV. Section V and Section VI are about the implementation and evaluation. In Section VII, some related works are introduced. Finally, we conclude our work in Section VIII.

## II. Background

### A. QR decomposition

QR decomposition, or QR factorization, is a matrix decomposition into a product of two matrices, see Equation 1.

$$A = QR. \tag{1}$$

After QR decomposition, given matrix $A$ will be decomposed into two matrices, $Q$ and $R$. The matrix $Q$ is an orthogonal matrix, which satisfies $Q^T = Q^{-1}$, or $Q^TQ = QQ^T = I$. The matrix $R$ is an upper triangular matrix, which has non-zero elements only at the upper right part, and zeroes for lower left part.

The QR decomposition is usually used to find solution of matrix equation, $Ax = b$, where $x$ and $b$ are column vectors. After QR decomposition, the equation can be written as the Equations 2 and 3.

$$QRx = b, \tag{2}$$

$$Q^TQRx = Rx = Q^Tb. \tag{3}$$

Here, we can find the solution $x$ easily, because $R$ is an upper triangular matrix.

**Algorithm 1** Householder QR decomposition

**procedure** HOUSEHOLDER QR
  $k \leftarrow 0$
  **while** $k < MatrixColSize$ **do**
    $\vec{a_k} \leftarrow (a_{1k}, ..., a_{nk})^T$
    $\vec{e_k} \leftarrow (0, ..., 0, 1, 0, ..., 0)^T$ ▷ 1 for k-th element
    $\alpha_k \leftarrow -sgn(a_{kk})||\vec{a_k}||$

    $\vec{u_k} \leftarrow \vec{a_k} + \alpha_k\vec{e_k}$
    $\vec{v_k} \leftarrow \vec{u_k}/||\vec{u_k}||$

    $Q_k = I - 2\vec{v_k}(\vec{v_k})^T$
    $A \leftarrow Q_kA$

    $k \leftarrow k + 1$
  **end while**

  $Q \leftarrow Q_1^TQ_2^T...Q_n^T$
  $R \leftarrow Q_nQ_{n-1}...Q_1A = Q^TA$
**end procedure**

One of the most useful methods for QR decomposition is the Householder reflections method, introduced in Algorithm 1 [4]. This algorithm uses orthogonal transformations to reflect the matrix columns through hyper planes: the Householder matrices. They are built from Householder reflectors, defining vectors of the projection. This algorithm computes the matrix column by column, starting from the left. Reflectors are computed and the column's under-diagonal elements reduced (zeroed) by the Householder matrix. The final matrix is the $R$ matrix, and then the $Q$ matrix can be calculated as a product of all Householder matrices needed to reach the result.

### B. Tiled QR decomposition

To parallelize QR decomposition, tiled QR decomposition methods are proposed [5], [6]. A given matrix is divided into several square or rectangle tiles, and a group of several tiles are processed on a processor or computing device, depending on the computing environment. For a tiled QR decomposition operation, there are four steps in the process.

*1) Triangulation:* In the triangulation step, also known as GEQRT, the QR decomposition operation is done for a single tile (Equation 4, with $t$ means a tile), and elements of the result matrix $R$ will substitute for elements of the processed tile (Equation 5, the notation $\leftarrow$ means replacement).

$$A_t = Q_tR_t, \tag{4}$$

$$A_t \leftarrow R_t. \tag{5}$$

*2) Update for triangulation:* The update for triangulation step is also known as UNMQR. After triangulation, the resulting matrix $Q$ should be multiplied by the right side tiles for the next operation (Equation 6).

$$A_t \leftarrow Q_tA_t. \tag{6}$$

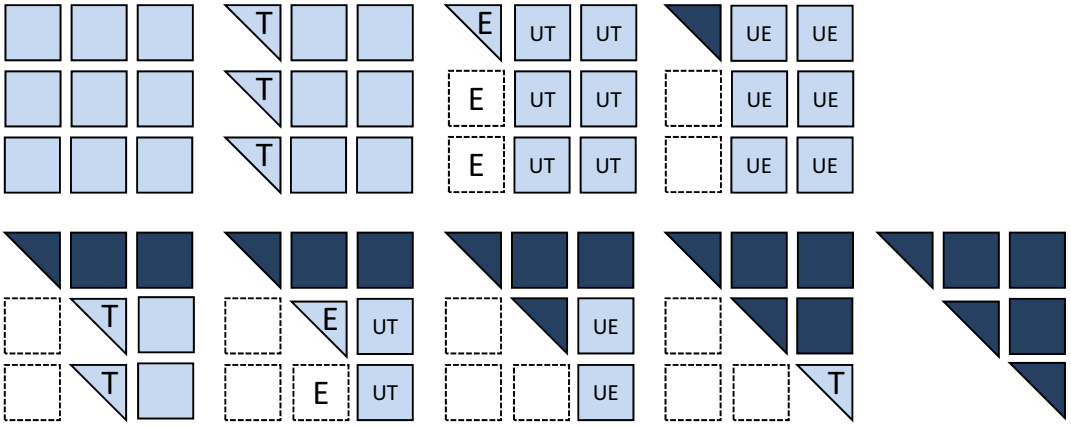



Fig. 2. Process of tiled QR decomposition of 3 by 3 tiles

*3) Elimination:* The elimination step is also known as TSQRT, or TTQRT. At the end of whole computation, the lower left part tiles should be zeroed, and in the elimination process it is done as shown in Equations 7 and 8 (the notations $t1$ and $t2$ mean two tiles on a same column). There are two types of elimination: triangle-on-top-of-triangle (TT) elimination and triangle-on-top-of-square (TS) elimination. TT elimination needs two triangulated tiles, and TS elimination needs one triangulated tiled and one non-triangulated tile. Both cases have same amount of arithmetic operation, and two tiles should exist on the same column of tiles.

$$\begin{bmatrix} A_{t1} \\ A_{t2} \end{bmatrix} = \begin{bmatrix} Q_{t1} & Q_{t2} \end{bmatrix} \begin{bmatrix} R_{t1} \\ 0 \end{bmatrix}, \tag{7}$$

$$\begin{bmatrix} A_{t1} \\ A_{t2} \end{bmatrix} \leftarrow \begin{bmatrix} R_{t1} \\ 0 \end{bmatrix}. \tag{8}$$

*4) Update for elimination:* After elimination, the result matrices $Q_1$ and $Q_2$ should be multiplied to the right side tiles for the next operation. This step is the update for elimination, which is also known as TSMQR or TTMQR, depending on the elimination process.

$$\begin{bmatrix} A_{t1} \\ A_{t2} \end{bmatrix} \leftarrow \begin{bmatrix} Q_{t1} & Q_{t2} \end{bmatrix} \begin{bmatrix} A_{t1} \\ A_{t2} \end{bmatrix}. \tag{9}$$

Fig. 3 shows a part of a directed acyclic graph (DAG) for the tiled QR operation. In the DAG, T, E, UT and UE denote triangulation, elimination, update for triangulation, and update for elimination, respectively. Each triangulation process leads the updating for rightward tiles and elimination for downward tiles, and each elimination process leads the updating for rightward tiles and triangulation for the next column tiles. Fig. 2 shows the tiled QR decomposition process for 3 by 3 tiles. The left-most column tiles are triangulated first, and elimination of the triangulated column and update for triangulation of the second and third columns are done. After the update for elimination of the second and third column tiles, the second column tiles will start triangulation, and there will be followed elimination of the second column and update for triangulation of the third column follows. This process will be repeated until all tiles finish the QR decomposition operation.

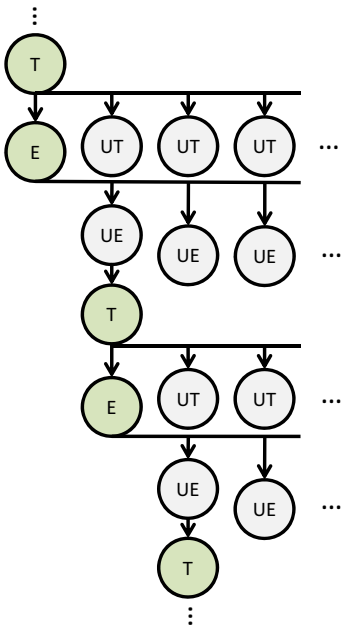
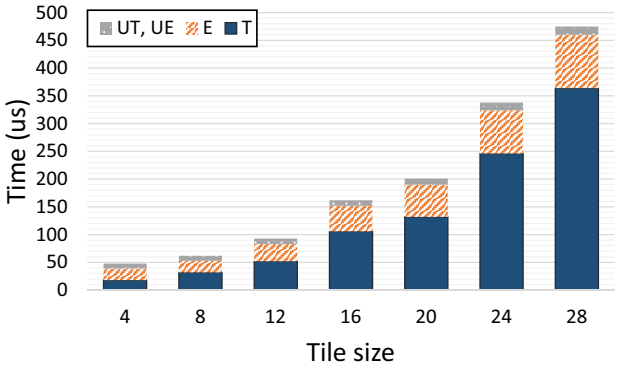



Fig. 3. A part of DAG of tiled QR decomposition

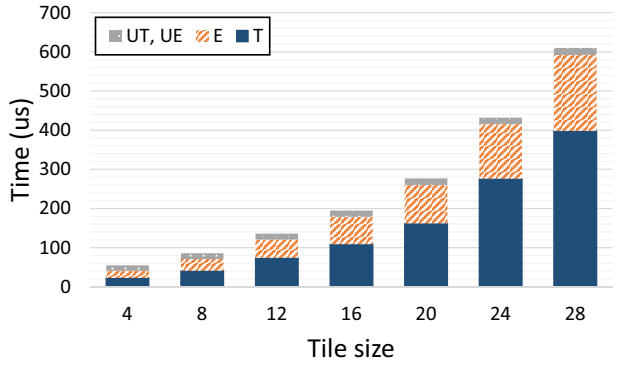
## III. Motivation

### A. Load change within each QR step

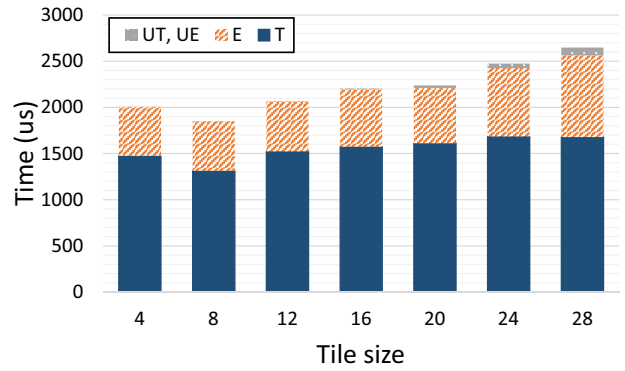
As we discussed in Section II, there are four steps for the tiled QR decomposition algorithm. In the algorithm, the characteristics of each step's operation are very different. Fig. 4 shows the operation time of each tiled QR decomposition step for a single tile on each device, and TABLE I shows the number of tiles to be operated, where the numbers of row and column tiles for the remaining part of the matrix are $M$ and $N$. The triangulation and elimination processes have more computation load than the other processes for a single tile,




(a) GTX580




(b) GTX680




(c) CPU (i7-3820, 4 cores)

Fig. 4. QR time for each step on each device

since they are based on an actual QR decomposition operation for each tile. In terms of parallelism, however, there are relatively few tiles that have to be triangulated or eliminated for each iteration. The two update processes have less load and higher parallelism. Therefore, we divide non-update processes and update processes depending on each device's computation speed and parallelism.

TABLE I
THE NUMBER OF TILES TO BE OPERATED FOR EACH STEP

| Step | Num. tiles |
|---|---|
| Triangulation | $M$ |
| Elimination | $M$ |
| Update for triangulation | $M \times (N-1)$ |
| Update for elimination | $M \times (N-1)$ |

(Remaining part $M$ by $N$)

### B. Heterogeneity of computing devices

Since each computing device has a different architecture, clock speed, memory bandwidth, and the number of parallel processors (or cores), the time taken for the QR operation should be different. For small matrix sizes or only a few tiles, a device that has a fast speed for each core can be effective for the entire operation, although the parallelism of the device is relatively low. For large matrix sizes or many tiles, a device that has high parallelism (a number of parallel cores) can be more powerful. In terms of tiled QR decomposition, triangulation and elimination are the cases of the former, and the two update processes are cases of the latter. In other words, there will be an appropriate device for each operation step in terms of performance and parallelism, and finding the appropriate device can increase the total performance.

### C. Effect of the number of devices

To maximize parallel operation, using as many computing devices as possible is trivial. For tiled QR decomposition, however, there are unavoidable data transfers among devices. After each triangulation and elimination process, matrices to update right-side tiles should be transferred among devices. If the number of devices increases, the total data size of the transfer will increase. This implies that there is a tradeoff between the parallelism (calculation speed) advantage and the data transfer overhead disadvantage.

This disadvantage is more critical for a small size matrix. Fig. 5 shows the proportion of calculation time and communication time, normalized by the total operation time, using four-core CPU and three GPUs. For small matrices, from 160 by 160 to 320 by 320, the portion of communication is more than 20 percent, and for larger matrices, the portion is less than 10 percent. This is because, for square matrices, if the matrix size increases, the number of tiles to be calculated increases as a square of the row or column size, and the number of tiles to be transferred increases proportionally to the row or column size.

Fig. 6 shows the time taken for various numbers of participating GPUs. Using a single GPU, two GPUs, and three GPUs are fastest on different sections of matrix sizes. The two tradeoff values vary depending on the number of tiles or the matrix size. Therefore, for a fixed matrix size, we can optimize the number of computing devices using the tradeoff terms, and using all available devices will not always give the best performance for some sizes of matrices.

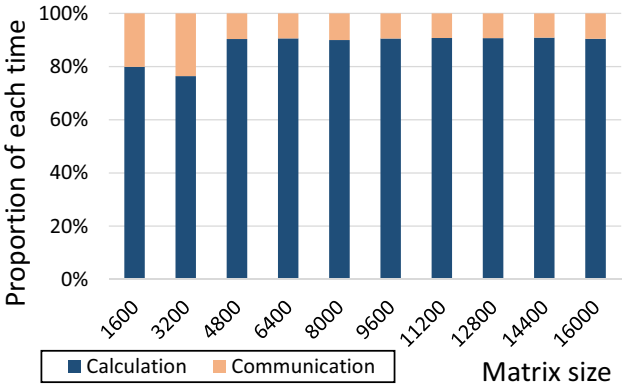



Fig. 5. Normalized calculation and communication time

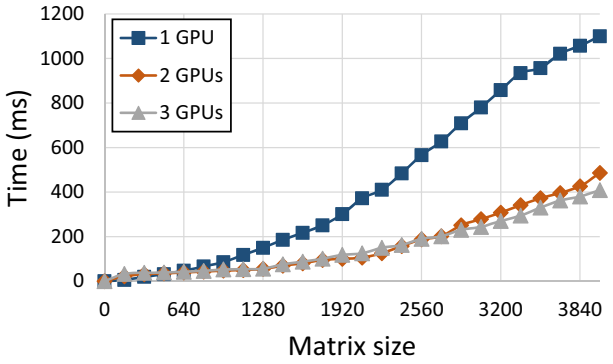
## IV. DESIGN

For a tiled QR operation, the tile and matrix shape can be square, tall and skinny, or short and fat. Each shape can be effective depending on the system. In our work, we use square tiles and focus on a square matrix. Some tiled QR algorithms, such as Song et al.'s work [7], use different tile sizes for CPUs and GPUs. If each tile is calculated on a fixed device, the tile size can be optimized depending on the device, and the size on each device may be different. In our work, however, some tiles can be calculated on multiple devices as will be discussed, so we use equal tile sizes for all devices. Here, load balancing is done depending on the number of distributed tiles, rather than the size of each tile.

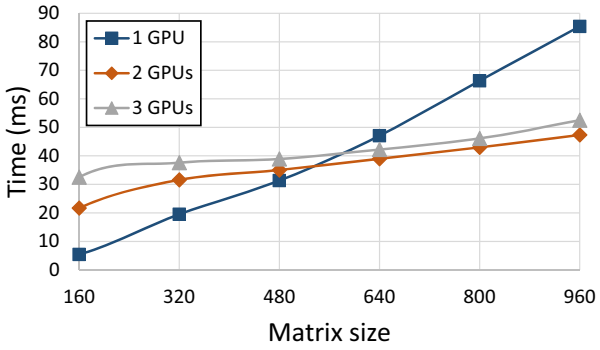
### A. Main computing device selection

The main computing device is the device that mainly executes the triangulation and elimination processes. Non-main devices receive the result $Q$ matrices of the main computing device, and they operate the update processes, which means their operations are dependent on the main computing device. Therefore, to maximize overall performance, the main computing device should be able to finish its job by the time the other devices finish. Of course, the main computing device can operate some of the update processes if the computation time on the main computing device is a lot faster than the others. If we select just the fastest device as a main computing device, the update processes should be done on a slower device, and the main computing device might wait too long for the other devices.

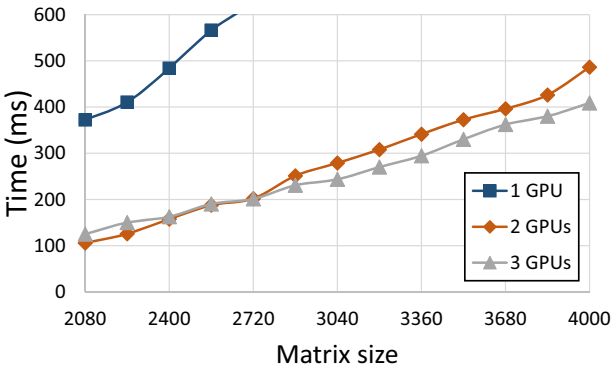
Synthetically, a device that is fast enough to finish the triangulation and elimination processes before the other devices' update processes will be selected as a main computing device, and the device doesn't have to be the fastest device. Algorithm 2 shows the selection process for the main computing device. It first finds candidate devices that can finish T or E before the UE or UT of other devices, and insert it into a list. After constructing the list, the device that has the minimum speed must be found, because non-minimum speed devices are better to be used to do update processes.




(a) Entire view




(b) Enlarged for size 160 to 960




(c) Enlarged for size 2,080 to 4,000

Fig. 6. Time taken for whole QR decomposition on various number of devices

### B. The number of devices selection

As we discussed in the Section III, if the number of devices increases, there will be a tradeoff between the parallelism advantage and the communication disadvantage. We try to find the optimal number of devices using the tradeoff factors. First, all devices are ranged in the descending order of the update processing speed, with the main computing device appearing first. Here, because the main computing device should participate in the QR decomposition algorithm,

**Algorithm 2** Main computing device selection

```
1: procedure MAIN_SELECTION
2:     for i ← 1 to numDev do
3:         if can_finish_T_before_UE() then
4:             if can_finish_E_before_UT() then
5:                 insert_list(i)
6:             end if
7:         end if
8:     end for
9:
10:    main ← find_minimum_speed_device_id()
11:    return main
12: end procedure
```

it comes at the head of the list. Then, for all available devices, we calculate the expected operation advantage and communication disadvantage for the first iteration. Since both times are proportional to the number of tiles, the trend for whole iteration will be similar to the first iteration.

The expected operation time $T_{op}(p)$ is determined by the Equation 10.

$$T_{op}(p) = \max_{1 \le i \le p} \begin{bmatrix} \#tile_m \times (time_m(T) + time_m(E)) \\ +\#tile(m) \times (time_m(UT) + time_m(UE)) \\ or \\ \#tile(i) \times (time_i(UT) + time_i(UE)) \end{bmatrix}. \quad (10)$$

Here, $\#tile(i)$ is the number of tiles distributed to the device $i$ for update processes, and $time_i(op)$ is the average time taken for single tile to do $op$, on the device $i$. The device $m$ denote the main computing device. $\#tile_m$ is the number of tiles for triangulation and elimination on the main computing device. The main computing device operates all four of the steps, so the time is calculated as the sum of the time taken for the four steps. For other devices, since they operate only update processes, the time will be a sum of the time taken for the update. Finally, because the operation itself is done in parallel on each device, the maximum value of operation time will finally determine $T_{op}$.

The expected communication time $T_{comm}(p)$ is determined by the Equation 11.

$$T_{comm}(p) = \sum_{i=1}^{p} \left( 3MT^2 \times size(element) \times \frac{1}{speed(m,p)} \right) + (M-1)T^2 \times size(element) \times \frac{1}{speed(j,m)}. \quad (11)$$

Here, $M$ is the number of row tiles and $T$ is the tile size. $size(element)$ is the size for each element of matrix, $speed(x, y)$ is the communication speed from $x$ to $y$, and $speed(x, y) = \infty$ if $x = y$. For the first iteration, $MT^2$ and $2MT^2$ should be transferred from the main computing device to other devices after triangulation and elimination respectively, for next update process. In the case of triangulation, each tile will generate one $Q$ matrix with size $T^2$, and the total size of these $Q$ matrices will be $MT^2$. For elimination, each couple tiles will generate two $Q$ matrices with size $T^2$, and the total size will be $2MT^2$. After elimination, the next column tiles, whose size is $(M-1)T^2$, will be transferred from the device $j$ to the main computing device, where $j$ is allocated device of next column.

Algorithm 3 shows the actual optimization process. If $N$ is the number of all available devices, the algorithm finds $p$ which minimizes $T(p) = T_{op}(p) + T_{comm}(p)$ for $1 \le p \le N$. After the $p$ value is finally determined, $p$ devices, which come from the head of the device list, participate in operating the update processes, including the main computing device.

**Algorithm 3** The number of devices selection

```
1: procedure NUM_DEV_SELECTION
2:     for i ← 1 to totalNumDev do
3:         insert_list(i)
4:     end for
5:
6:     order_list_by_update_speed()
7:     move_main_dev_to_list_head()
8:
9:     T(0) ← ∞
10:    p_min ← 0
11:    for p ← 1 to list_size do
12:        T_op(p) ← CALC_T_op(p)
13:        T_comm(p) ← CALC_T_comm(p)
14:        T(p) ← T_op(p) + T_comm(p)
15:
16:        if T(p) < T(p_min) then
17:            p_min ← p
18:        end if
19:    end for
20:    return p_min
21: end procedure
22:
23:
24: procedure CALC_T_op(p)
25:    return max_{1≤i≤p}
26:        [ #tile_m × (time_m(T) + time_m(E))
           +#tile(m) × (time_m(UT) + time_m(UE))
                        or
           #tile(i) × (time_i(UT) + time_i(UE)) ]
27: end procedure
28:
29:
30: procedure CALC_T_comm(p)
31:    return Σ_{i=1}^{p} ( 3MT^2 × size(element) × 1/speed(c,p) )
32:    +(M−1)T^2 × size(element) × 1/speed(j,c)
33: end procedure
```

### C. Tile distribution

After dividing a given matrix, there will be many tiles. Grouping those tiles for each column, the column tiles will be

distributed into $p$ devices. Here, $p$ is the value from the number of devices in the optimization process. Algorithm 4 shows how to distribute tiles in our work. Before distribution, we first find the integer ratio of all the devices using the number of tiles that can be updated in a unit time. For example, if three devices, whose IDs are 0, 1 and 2, can process 8, 12 and 4 tiles in a unit time, respectively, the ratio will be $2:3:1$.

Using this ratio, we can construct a distribution guide array, which is a kind of cyclic string array. The length of the array is the sum of all the ratio values. Within the array, the ID of each device is cyclically distributed, depending on distributed according to the ratio that we calculated. While the whole array is generated, the algorithm will find the maximum ratio value, insert the ID of that ratio into the array, and reduce the ratio value by 1. For example, if the ratio of three devices is $2:3:1$, the distribution guide array will be $\{1,0,1,0,1,2\}$. When the number of columns is not a multiple of the length of the array, the last few columns may not be distributed as the ratio. Since more tiles on a faster device is better for performance, we construct the array with the device with a larger ratio value appearing first.

Finally, according to the array, the $i$-th column tiles are distributed by the Equation 12, except the first column tiles distributed to the main computing device, because their only operations are triangulation and elimination.

$$distribute(i) = guide\_array[i\%length(guide\_array)]. \quad (12)$$

### D. Tiled QR decomposition progress

The actual process of tiled QR decomposition is the same as introduced one in Section II. As already shown in Fig. 3, triangulation comes first, elimination and update for triangulation follow, and next triangulation begins with the update for the elimination process. At the end of each process, appropriate data transfer is needed, depending on our distribution method.

After triangulation, update matrices are transferred from the main computing device to non-main devices. After elimination, update matrices are transferred as in the triangulation case, and the next column tiles are also transferred from one non-main devices to the main computing device for next triangulation process. This iteration is done until all tiles finish their QR operation.

## V. Implementation

Fig. 7 shows the tiled QR decomposition system structure. The manager thread, which runs on a CPU, selects the main computing device, decides the number of participating devices, distributes tiles, and migrates dependent data among the devices. Computing threads, which also run on a CPU, will do appropriate computations using the allocated device, CPU or GPU. Each computing thread for CPU can have several CPU slave threads which operate jobs in parallel. Each computing thread for a GPU can have a single GPU, and the computation threads of the GPU will behave as slave threads in the CPU case.

**Algorithm 4** Tile distribution with guide array

```
1:  procedure GET_RATIO(i)
2:      get_number_of_tile_update_on_unit_time()
3:      get_integer_ratio()
4:      return integer_ratio_value
5:  end procedure
6:
7:
8:  procedure GENERATE_ARRAY
9:      ratio_sum ← 0
10:     for i ← 1 to p do
11:         ratio(i) ← GET_RATIO(i)
12:         ratio_sum ← ratio_sum + ratio(i)
13:     end for
14:
15:     for i ← 0 to ratio_sum do
16:         id ← find_maximum_ratio_value()
17:         guide_array[i] ← id
18:         ratio(id) ← ratio(id) − 1
19:     end for
20:     return guide_array
21: end procedure
22:
23:
24: procedure ALLOCATE(index, id)
25:     The column at [index]
26:     will be allocated to the device[id]
27: end procedure
28:
29:
30: procedure DISTRIBUTION
31:     guide_array ← GENERATE_ARRAY()
32:     array_length ← length(guide_array)
33:
34:     ALLOCATE(1, main_dev)
35:     for i ← 2 to p do
36:         ALLOCATE(i, guide_array[i%array_length])
37:     end for
38: end procedure
```

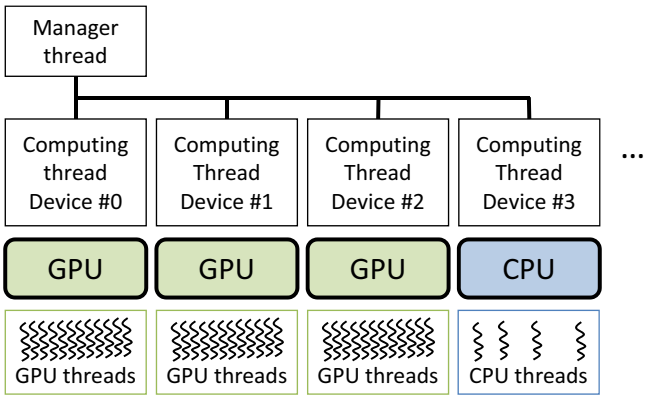



Fig. 7. CPU and GPU heterogeneous computing system structure

TABLE II
EVALUATION ENVIRONMENT

| | |
|---|---|
| CPU | Intel i7-3820 (Quad core, 3.6GHz) |
| Main Memory | 32GB |
| GPU | Two GTX680 (1536 cores)<br>One GTX580 (512 cores) |
| OS | Ubuntu 12.04 |
| Linux version | 3.2.0 |
| GPU driver version | 304.54 |
| CUDA version | 5.0 |
| Compiler | g++ 4.6.3 and nvcc 5.0 v0.2.1221 |

CPU processes are implemented based on the PLASMA library [8]. At the beginning, each computing thread for the CPU gets ready to use the library. For the GPU part, we implement QR decomposition based on the Householder algorithm [4]. As a tile size, we use 16 by 16 because the number of cores of the CPU and GPUs are the power of 2. For the element data, we generate random floating point numbers.

## VI. EVALUATION

For evaluation hardware, we construct the single node computing environment as TABLE II. We use four core CPU and three GPUs. Two GPUs are GTX680, which has 1536 cores. The other GPU is GTX580, which has 512 cores.

For software, we use Ubuntu 12.04 OS, with Linux kernel 3.2.0. The version of the GPU driver is the NVIDIA graphic driver 304.54, and the CUDA version is 5.0. As compilers, g++ 4.6.3 and nvcc 5.0 v0.2.1221 are used.

### A. Scalability

First, we evaluate scalability. Fig. 8 shows the QR decomposition time taken for various numbers of computing devices, with changing the matrix size. Because the number of cores of each device is different, we arrange the experiment data as the number of parallel cores of the devices used. The x-axis is the number of parallel cores in a logarithmic scale. The four points of each graph are for using only a CPU with 4 cores, a CPU with one GTX580, a CPU with one GTX580 and one GTX680, and a CPU with all available GPUs, respectively. The y-axis is the time taken for the whole QR decomposition operation as a unit of a second, also in a logarithmic scale. In the figure, each graph shows decreasing behavior for all five matrix sizes. For a 3,200 by 3,200 matrix, we can reduce the operation time from 19.9 seconds to 0.28 seconds. For a 6,400 by 6,400 matrix, we can reduce the time from 73.5 seconds to 1.09 seconds. For other cases, 9,600 by 9,600, 12,800 by 12,800, and 16,000 by 16,000 matrices, we can reduce the time from 171.7, 269.3, and 462.1 seconds to 2.52, 4.24, and 6.87 seconds, respectively. Here, because all graphs are proportionally decreasing, we can conclude that our method has nice scalability.

### B. Main computing device selection

Next, we evaluate whether the selected main computing device can maximize the performance or not. In our environment, there are three types of computing devices: CPU, GPU-GTX580, and GPU-GTX680. The processing speed for each

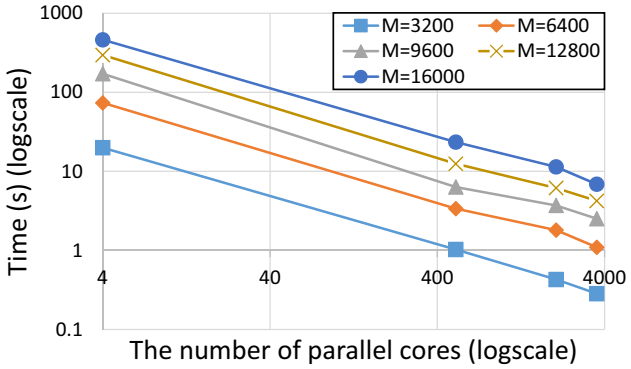



Fig. 8. Scalability: (a) CPU 4 cores, (b) CPU+1GPU 516 cores, (c) CPU+2GPUs 2,052 cores, (d) CPU+3GPUs 3,588 cores

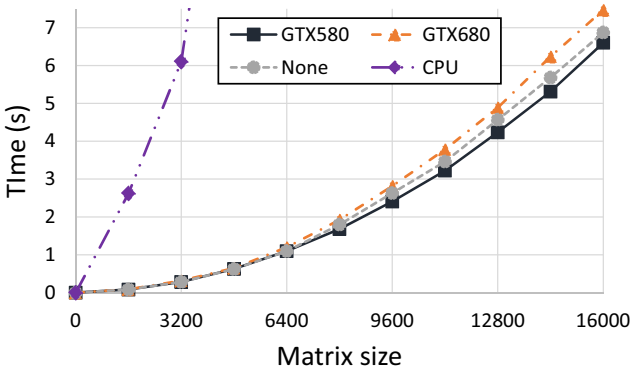



Fig. 9. Time taken depending on the main computing device selection

device is already shown in Fig. 4. Because the triangulation and elimination speed of the CPU is too slow compared to other devices' update speed, it is not good to use the CPU as the main computing device. In case of GTX680, the processing speed is slower than GTX580, but it has more parallel cores, 1,536, than GTX580's 512 cores. Here, using GTX680 for the update processes is more useful than using it as a main computing device. Therefore, our selection is GTX580.

Fig. 9 shows the QR decomposition time taken for various selections of the main computing device. The x-axis is the matrix row and column sizes. The y-axis is the time taken for the whole QR decomposition operation as a unit of a second. There are four cases, GTX580 as the main computing device (our selection), GTX680 as the main, no specific main computing device in which all GPUs process their own triangulation and elimination, and CPU as the main. The last case, using the CPU as the main computing device, is not shown well in the figure, because it is too slow: 6.1, 26.3, 76.4, 205.7, 430.6 seconds for 3,200, 6,400, 9,600, 12,800, 16,000 matrix sizes, respectively. For appropriate selection, in the case of 16,000 by 16,000 matrix, we can speed up 13 percent from selecting the main as another GPU, and 5 percent from not selecting a specific main computing device.

TABLE III
THE NUMBER OF DEVICES OPTIMIZATION

| | Predicted | | | Actual | | |
|---|---|---|---|---|---|---|
| Matrix size | 1 G | 2 G | 3 G | 1 G | 2 G | 3 G |
| 160 | **1.00** | 3.73 | 5.27 | **1.00** | 4.00 | 5.99 |
| 320 | **1.00** | 2.77 | 3.82 | **1.00** | 1.62 | 1.93 |
| 480 | **1.00** | 1.40 | 2.11 | **1.00** | 1.12 | 1.24 |
| 640 | 1.02 | **1.00** | 1.43 | 1.21 | **1.00** | 1.08 |
| 800 | 1.03 | **1.00** | 1.34 | 1.54 | **1.00** | 1.07 |
| 960 | 1.03 | **1.00** | 1.29 | 1.80 | **1.00** | 1.11 |
| 1120 | 1.06 | **1.00** | 1.23 | 2.35 | **1.00** | 1.07 |
| 1280 | 1.09 | **1.00** | 1.19 | 2.71 | **1.00** | 1.02 |
| 1440 | 1.13 | **1.00** | 1.16 | 2.68 | **1.00** | 1.09 |
| 1600 | 1.15 | **1.00** | 1.13 | 2.71 | **1.00** | 1.09 |
| 1760 | 1.18 | **1.00** | 1.10 | 2.63 | **1.00** | 1.06 |
| 1920 | 1.20 | **1.00** | 1.08 | 3.00 | **1.00** | 1.18 |
| 2080 | 1.22 | **1.00** | 1.05 | 3.52 | **1.00** | 1.18 |
| 2240 | 1.23 | **1.00** | 1.04 | 3.27 | **1.00** | 1.19 |
| 2400 | 1.25 | **1.00** | 1.02 | 3.07 | **1.00** | 1.03 |
| 2560 | 1.26 | **1.00** | 1.01 | 3.02 | **1.00** | 1.01 |
| 2720 | 1.28 | 1.01 | **1.00** | 3.12 | 1.01 | **1.00** |
| 2880 | 1.31 | 1.02 | **1.00** | 3.07 | 1.09 | **1.00** |
| 3040 | 1.34 | 1.03 | **1.00** | 3.20 | 1.14 | **1.00** |
| 3200 | 1.36 | 1.04 | **1.00** | 3.18 | 1.14 | **1.00** |
| 3360 | 1.38 | 1.05 | **1.00** | 3.17 | 1.16 | **1.00** |
| 3520 | 1.40 | 1.07 | **1.00** | 2.90 | 1.13 | **1.00** |
| 3680 | 1.42 | 1.07 | **1.00** | 2.82 | 1.09 | **1.00** |
| 3840 | 1.44 | 1.08 | **1.00** | 2.78 | 1.12 | **1.00** |
| 4000 | 1.46 | 1.09 | **1.00** | 2.69 | 1.19 | **1.00** |

(*) Normalized value for smallest time

### C. The number of devices

To verify the optimization of the number of devices, we compare the result from the predicted number and the actual fastest number of devices. We only consider the number of GPUs, because the aid of the CPU is not much effective for determining the number of devices. TABLE III shows the predicted time, which is $T_{op} + T_{comm}$, and the actual computation time, with normalization for the smallest time value for each case. The notation 1G, 2G and 3G means using one GPU (GTX580), two GPUs (GTX580 and 680), and all three GPUs, respectively. Because we selected GTX580 as the main computing device, it appears as the first of the device list. For matrix sizes 160 by 160 to 480 by 480, using only a single GPU is the fastest one among the three cases for both prediction and actual result. For matrix sizes from 640 by 640 to 2,560 to 2,560, using two GPUs is faster than a single GPU or three GPUs, of course for both results. Finally, for matrix sizes larger than 2,720 by 2,720, using all three GPUs the fastest. Through all matrix sizes, the number of devices, which is predicted to have a minimal $T_{op} + T_{comm}$ value, can maximize actual performance. Therefore, the suggested equations can actually optimize the operation time.

### D. Tile distribution

Now, we evaluate the tile distribution with the distribution guide array. We construct a distribution guide array based on the number of tiles that can be processed in a unit time for each device. Fig. 10 shows the QR decomposition time for the various distribution methods. The x-axis is the matrix row and column size, and the y-axis is the time taken for the operation

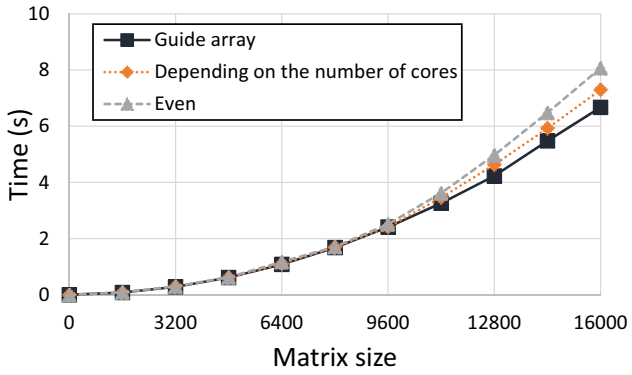



Fig. 10. Time taken depending on the tile distribution

in a unit of a second. We compare three distributions: using the distribution guide array (our method), distribution based on the number of cores of each device, and the same number of tiles distribution for GPUs with some tiles on the GPU depending on the number of cores. For smaller matrix sizes, the distribution method does not have much effect, because the actual distribution itself is not much different. As the matrix size becomes larger, each method shows different increasing speed. For a 16,000 by 16,000 matrix, our method is 21 percent faster than the evenly distribution method, and 10 percent faster than the number of cores based method.

## VII. RELATED WORKS

There have been some attempts to compute QR decomposition efficiently on the CPU and GPU hybrid core system. Early attempts [9], [10] tried to run QR decomposition on the GPU, with porting existing parallel QR libraries into the GPU. Since they tried to do almost all the calculations on the GPU to avoid communication, they had low CPU utilization. In our work, we used all the computing devices which participate in the QR decomposition.

Agullo et al. [11] used the dynamic tile migration method between the CPU and GPU. During program construction, the programmer can select parts of the code which might run on the GPU, and the actual operation device is determined on the runtime. This algorithm can select the appropriate device at the runtime, with accompanying device monitoring overhead. Our work focused on tile migration for specific tiles at a specific time.

Communication-Avoiding QR algorithms [12], [13] distribute tiles to minimize the communication cost. They divide the matrix into row by row and the group row tiles are distributed into a single cluster, where their target system is a multi-cluster system. However, in our work, we use a column by column tile distribution, which is easy for load balancing, since there is not much communication cost for our system.

Song et al. [7] proposed an auto tuning method. They first ran a small size of matrix on the system to find the most appropriate tile size for each computing device, CPU and GPU, and adapted the tuned tile size into a larger matrix.

In our work, we tried to optimize more mathematically, and we adjusted the number of fixed size tiles rather than the tile size itself for load balancing.

## VIII. Conclusion

In this paper, we introduced mathematical optimization for tiled a QR decomposition operation on a CPU and GPU heterogeneous computing system. We first divided the single tile operation into four steps. We selected a specific device as the main computing device, which mainly operates the triangulation and elimination process. We also optimized the number of devices that would participate. For all available devices, we can find the optimal number based on the tradeoff between the operation time and the communication cost. Additionally, we introduced the distribution guide array, which can distribute tiles for update processes efficiently to several devices.

As a further work, QR decomposition of very large matrix can be considered. Our current work assumes that there is no problem about memory size, while a lack of memory problem can occur for very large matrix sizes. Another one is expanding the suggested equations, which is verified for CPU and GPU heterogeneous cores, into other computing devices, or a multi node environment.

## References


[1] Intel. The intel xeon phi coprocessor. [Online]. Available: http://www.intel.com/content/www/us/en/high-performance-computing/high-performance-xeon-phi-coprocessor-brief.html

[2] NVIDIA. Cuda official cite. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

[3] Cuda. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

[4] A. S. Householder, "Unitary triangularization of a nonsymmetric matrix," *J. ACM*, vol. 5, no. 4, pp. 339–342, Oct. 1958. [Online]. Available: http://doi.acm.org/10.1145/320941.320947

[5] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, "A class of parallel tiled linear algebra algorithms for multicore architectures," *Parallel Computing*, vol. 35, no. 1, pp. 38 – 53, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819108001117

[6] H. Bouwmeester, M. Jacquelin, J. Langou, and Y. Robert, "Tiled qr factorization algorithms," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, nov. 2011, pp. 1 –11.

[7] F. Song, S. Tomov, and J. Dongarra, "Enabling and scaling matrix computations on heterogeneous multi-core and multi-gpu systems," in *Proceedings of the 26th ACM international conference on Supercomputing*, ser. ICS '12. New York, NY, USA: ACM, 2012, pp. 365–376. [Online]. Available: http://doi.acm.org/10.1145/2304576.2304625

[8] Plasma library. [Online]. Available: http://icl.cs.utk.edu/plasma/software/index.html

[9] M. Fogue, F. D. Igual, E. S. Quintana-Orti, and R. A. van de Geijn, "Retargeting plapack to clusters with hardware accelerators," in *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, 28 2010-july 2 2010, pp. 444 –451.

[10] G. Quintana-Ortí, F. D. Igual, E. S. Quintana-Ortí, and R. A. van de Geijn, "Solving dense linear systems on platforms with multiple hardware accelerators," in *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*, ser. PPoPP '09. New York, NY, USA: ACM, 2009, pp. 121–130. [Online]. Available: http://doi.acm.org/10.1145/1504176.1504196

[11] E. Agullo, C. Augonnet, J. Dongarra, M. Faverge, H. Ltaief, S. Thibault, and S. Tomov, "Qr factorization on a multicore node enhanced with multiple gpu accelerators," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, may 2011, pp. 932 –943.

[12] F. Song, H. Ltaief, B. Hadri, and J. Dongarra, "Scalable tile communication-avoiding qr factorization on multicore cluster systems," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11. [Online]. Available: http://dx.doi.org/10.1109/SC.2010.48

[13] M. Anderson, G. Ballard, J. Demmel, and K. Keutzer, "Communication-avoiding qr decomposition for gpus," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, may 2011, pp. 48 –58.