The performance of the GPU-accelerated ELPA2 solver is tested on the Ascent computer at Oak Ridge National Laboratory. Each node of Ascent has 2 IBM POWER9 CPUs with 42 cores in total, and 6 NVIDIA Volta GV100 GPUs. As Fig. 1 shows, the bottlenecks seem to be the first and fourth steps.

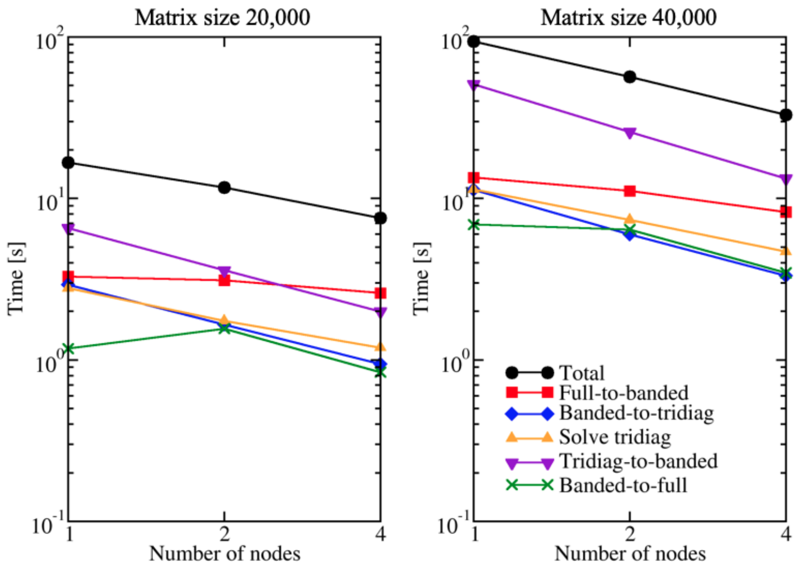

Figure 1: Time to solution of the five computational steps of ELPA2 with respect to the number of nodes. Matrix sizes are 20,000 $\times$ 20,000 (left) and 40,000 $\times$ 40,000 (right). Tests are performed on the Ascent computer at the Oak Ridge National Laboratory. Each node consists of 2 IBM POWER9 CPUs and 6 NVIDIA Volta GV100 GPUs. The nodes are fully exploited by running 42 MPI tasks and 6 GPUs per node.

This document describes a CUDA kernel for applying a series of Householder transformations to a matrix, which is the main computational task in the fourth step of ELPA2. Mathematically this is as simple as

$$\boldsymbol{Y} = \boldsymbol{H}_1 \boldsymbol{H}_2 \cdots \boldsymbol{H}_n \boldsymbol{X}. \tag{1}$$

$\boldsymbol{H}_i$ is a Householder transformation matrix taking the form

$$\boldsymbol{H}_i = \boldsymbol{I} - \tau_i \boldsymbol{v}_i \boldsymbol{v}_i^*, \tag{2}$$

with $\boldsymbol{\tau}_i$ being a scalar, and $\boldsymbol{v}_i$ a vector. Since a Householder matrix can be completely defined by $\boldsymbol{\tau}_i$ and $\boldsymbol{v}_i$, there is no need to explicitly construct and store $\boldsymbol{H}_i$. Instead, only $\boldsymbol{\tau}_i$ and $\boldsymbol{v}_i$ are stored.

In an actual ELPA2 calculation with MPI, each MPI process applies n Householder transformations to the local portion of the eigenvector matrix $\boldsymbol{X}$. The length of each Householder vector $\boldsymbol{v}_i$ is b, which is always equal to the semi-bandwidth nbw of the banded matrix stage in ELPA2. The dimension of the local eigenvector matrix is $N_R \times N_C$, where the number of rows $N_R = b + n - 1$, and the number of columns N_C is a user-specified parameter (usually 1,024).

The n Householder transformations must be transformed one after another. This is done in a loop:

```
do i = n,-1,1
  apply the n^th transformation
end do
```

Fig. 2 shows the data layout of the Householder transformation CUDA kernel, using the n^{th} and $(n-1)^{th}$ iterations as an example. Here $b = 4$, $n = 4$, $N_R = b + n - 1 = 7$, and $N_C = 6$. In the n^{th} iteration, the n^{th} Householder vector is applied to the eigenvector matrix $\boldsymbol{X}$. Note that the n^{th} $\boldsymbol{v}$ only alters the last 4 rows of $\boldsymbol{X}$ (yellow part in Fig. 2). This is a general property of Householder transformations. The kernel is launched with N_C blocks and b threads, such that each block works on one column of the eigenvector matrix, whereas each thread within a block in turn works on one element of the eigenvector matrix. For each block, the task is to compute

$$\boldsymbol{H}\boldsymbol{x} = (\boldsymbol{I} - \tau\boldsymbol{v}\boldsymbol{v}^*)\boldsymbol{x} = \boldsymbol{x} - \tau\boldsymbol{v}(\boldsymbol{v}^*\boldsymbol{x}). \tag{3}$$

Again, the scalar τ and vector $\boldsymbol{v}$ are used to define a Householder transformation. The vector $\boldsymbol{x}$ is a sub-vector of the eigenvector matrix $\boldsymbol{X}$. The lengths of $\boldsymbol{v}$ and $\boldsymbol{x}$ are both equal to b. Eq. 3 is computed in two steps. First, the dot product $d = \boldsymbol{v}^*\boldsymbol{x}$ is computed by multiplying the corresponding element of $\boldsymbol{v}$ and $\boldsymbol{x}$ on each thread, then performing a parallel reduction within a block. Second, the vector $\boldsymbol{x}$ is updated by $\boldsymbol{x} - \tau\boldsymbol{v}(\boldsymbol{v}^*\boldsymbol{x})$, where the threads in a block can work independently from each other.

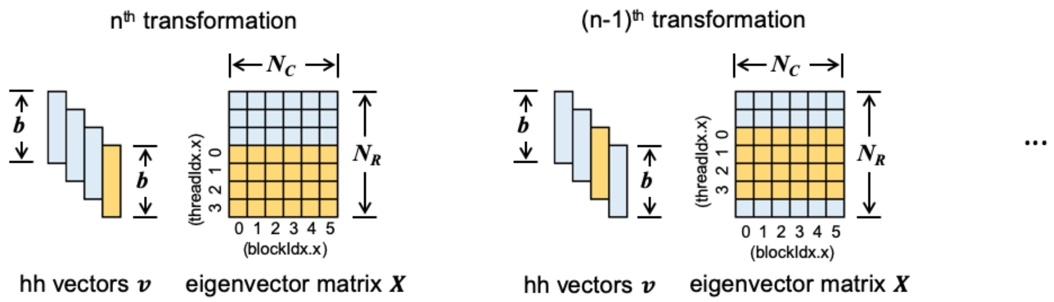

Figure 2: Workflow of the Householder transformation CUDA kernel. $b = 4$, $n = 4$, $N_R = b+n-1 = 7$, and $N_C = 6$. A block works independently on a column of the eigenvector matrix. A thread works on one element of the eigenvector matrix. From the n^{th} iteration to the $(n-1)^{th}$ iteration, the working part of the eigenvector matrix is shifted upward by one element.

When the n^{th} transformation is finished, the last row of the eigenvector matrix $\boldsymbol{X}$ will no longer be updated. Therefore, within every block, the thread that currently holds the last row of $\boldsymbol{X}$ writes the value to the result matrix. Then in the $(n-1)^{th}$ transformation, thread t, $t > 0$, takes the value of thread $t-1$, whereas thread $t == 0$ reads in a new row from the eigenvector matrix $\boldsymbol{X}$. The computation of Eq. 3 can proceed as before. The kernel finishes when all the n Householder transformations are applied to the eigenvector matrix.