Supplementary of X-TED

1 PSEUDOCODE OF DYNAMIC PARALLEL STRATEGY

Based on the approach we discuessed in Section 6, the dynamic parallel strategy is proposed and described in Algorithm 1. The algorithm iterates all depths from 0 to the maximum depth, processing all tables with the current depth in parallel. These tables are categorized into four types according to their size so that different approaches can be adopted for tables in different groups. Three thresholds are used as criteria in the table categorization process. Each approach has been implemented, and the procedure for fetching tables at the current depth is also executed in parallel on GPU.

Algorithm 1 Dynamic parallel strategy on GPU

Input: Small tables queue $small$, Medium tables queue $medium$
Input: Large tables queue $large$, Huge tables queue $huge$
Input: Threshold $\theta_1, \theta_2, \theta_3$
Input: Max table depth max, Depths of tables (td), Sizes of tables (ts)
Output: Edit distance results computation on GPUs

procedure DYNAMIC-PARALLEL
  for $depth \leftarrow 0$ **to** max **do**
    FETCH-TASKS($depth, small, medium, large, huge, td, ts$)
    ▷ Dynamically use different approaches for different size-type of tables
    SINGLE-THREAD-COMPUTING($small$)
    SINGLE-WARP-COMPUTING($medium$)
    SINGLE-BLOCK-COMPUTING($large$)
    MULTI-BLOCKS-COMPUTING($huge$)
  end for
end procedure

procedure FETCH-TASKS
  ▷ Find all tables within the current depth and categorize them
  for $i \leftarrow 1$ **to** $td.row()$ **do**
    for $j \leftarrow 1$ **to** $td.column()$ **do**
      if $depth = td[i][j]$ **then**
        if $ts[i][j] \leq \theta_1$ **then**
          $small.append(table_{i,j})$
        else if $ts[i][j] \geq \theta_1 \wedge ts[i][j] \leq \theta_2$ **then**
          $medium.append(table_{i,j})$
        else if $ts[i][j] \geq \theta_2 \wedge ts[i][j] \leq \theta_3$ **then**
          $large.append(table_{i,j})$
        else
          $huge.append(table_{i,j})$
        end if
      end if
    end for
  end for
end procedure

In the multi-block approach, a well-designed scheduling mechanism shown in Algorithm 2 allows all blocks to collaborate in computing tables while continuously fetching tasks for computation, resulting in more efficient resource utilization. The required number of blocks for each table is determined based on their size before computing. As many as tables are then efficiently distributed among the available blocks on the GPU. Once a table's computation is completed, the free blocks seamlessly transition to working on another table until all tables are successfully computed.

Algorithm 2 Scheduling in multi-blocks parallel computing

Input: Huge tables queue $huge$, Blocks needed for each huge table $blks$
Output: Computation for huge tables on GPUs

procedure MULTI-BLOCKS-COMPUTING
  ▷ Initialize the beginning and ending block for the first table
  $begin \leftarrow 0$
  $end \leftarrow blks[1] - 1$
  for $i \leftarrow 1$ **to** $huge.size()$ **do**
    if $blockId \in [begin, end]$ **then**
      ▷ Use these blocks to compute current table cooperatively
      COOPERATIVE-COMPUTING($huge[i], blockId, begin, end$)
    end if
    if $end + blks[i+1] \leq numBlocks$ **then**
      $begin \leftarrow end + 1$
      $end \leftarrow end + blks[i+1]$
    else
      $begin \leftarrow 0$
      $end \leftarrow blks[i+1] - 1$
    end if
  end for
end procedure

2 DETAILS FOR MATHEMATICS MODEL

As mentioned in Section 6.2, X-TED relies on three crucial thresholds to determine the approaches applied in TED table computing. Therefore, accurately finding the values of these thresholds is of utmost importance as it directly impacts the algorithm's performance. In this section, we present the explanations of our mathematics model that serves as a guide to determine the appropriate values for these thresholds, thereby enabling X-TED to achieve optimal performance.

Assume the computing time for each unit in a table is identical and denote it as t_u, for a table with n rows and n columns, because the single thread traverses and computes all units, the computation time using the single-thread method can be calculated as follows:

$$T_{thread} = n^2 \cdot t_u \tag{1}$$

In the single-warp computing, all units are processed in the parallel wavefront manner. If the warp size w is larger than the number of rows n, all units in the table can be processed in one iteration. The number of waves in total is $2n - 1$, and within each wave, the computation cost includes the time of unit along with the overhead of warp-level synchronization. On the other hand, if the warp size is smaller than n, the warp needs to move to the top after finishing the bottom area until all units are calculated. Consequently, the computing time for the single-warp approach is as presented below:

$$T_{warp} = \begin{cases} (2n-1) \cdot (t_u + sync_{warp}) & n \leq w \\ \lceil \frac{n}{w} \rceil \cdot (w + n - 1) \cdot (t_u + sync_{warp}) & n > w \end{cases} \tag{2}$$

In the third method, referred to as single-block computing, the mechanism remains similar, and the block size is denoted as b. The difference is that all threads in the same block engage in block-level synchronizations, which is more time-consuming compared to the warp-level approach. Therefore, the computing time in this method

can be modeled as the following formula:

$$T_{block} = \begin{cases} (2n-1)\cdot(t_u + sync_{block}) & n \le b \\ \lceil \frac{n}{b} \rceil \cdot (b+n-1)\cdot(t_u + sync_{block}) & n > b \end{cases} \quad (3)$$

The final approach is multi-blocks computing. In this approach, we assume that a total of k blocks are used to compute one table, and the inter-block synchronization for these multiple k blocks is marked as $sync_m(k)$. If the number of threads $(k \cdot b)$ within these blocks is greater than the number of table rows, all units can be calculated in one iteration by the wavefront pattern and the number of waves is thus $(k \cdot b + n - 1)$. Otherwise, it still requires several iterations to gradually move to the top and compute all data. The computation time can be calculated as below:

$$T_{mblocks} = \begin{cases} (2n-1)(t_u + sync_m(k)) & n \le k \cdot b \\ \lceil \frac{n}{k \cdot b} \rceil (k \cdot b + n - 1)(t_u + sync_m(k)) & n > k \cdot b \end{cases} \quad (4)$$

The four mathematical formulas above constitute a comprehensive model designed to determine the thresholds effectively. As the table size $(n \times n)$ changes, these formulas generate four distinct curves, representing the time taken for each parallel approach. The threshold θ_1, which marks the transition from the single-thread method to the single-warp approach, is the point where the curves of Formula (1) and Formula (2) intersect. Also, threshold θ_2 indicates the switch from the single-warp to the single-block computing, which occurs at the intersection point between the curves of Formula (2) and Formula (3). Similarly, the last threshold, threshold θ_3, is the intersection between the curves of Formula (3) and Formula (4).

From these formulas, it is evident that the four machine-dependent variables (t_{unit}, $sync_{warp}$, $sync_{block}$, and $sync_m(k)$) determine the curves and their intersection points. For any given GPU device, the overheads for $sync_{warp}$, $sync_{block}$, and $sync_m(k)$ remain fixed. Additionally, the computing time for a single unit (t_{unit}) in a TED table is constant and can be easily tested by calculating the average unit time when computing a small table. Once the values of these four variables are obtained, calculating the three intersection points and determining the thresholds (θ_1, θ_2, and θ_3) becomes straightforward. Therefore, for this specific GPU machine, these three thresholds can be applied in any other extensive TED computations to achieve optimal performance.

We have tested the validity of this mathematics model on our GPU machine. Each $sync_{warp}$ consumes 0.24 microseconds on average, each $sync_{block}$ takes 0.63 microseconds on average, and when the k is set to 2, the time for $sync_m(k)$ is 1.64 microseconds on average. Also, we tested the time for computing a single unit is approximately 0.19 microseconds on average. The warp size w is set to 32 and the block size b is set to 256. Therefore, according to the Formula (1) and Formula (2), the n value of intersection point is 3.95, which means that the threshold θ_1 of table size is nearly 16. By the Formula (2) and Formula (3), it can be calculated that the intersection point is when n is 90.37, and consequently the value of θ_2 should be approximately 8167. Similarly, the curves of Formula (3) and Formula (4) intersects when n is 886.98, and the θ_3 for table size should be around 786734.

The actual intersection points between these methods are presented in Figure 1. For each approach, we have tested the averaged time for computing one table over the change of table size in 50 runs. The actual intersection point θ_1 between the single-thread and single-warp computing is nearly 25. And the actual values of θ_2 is around 8836 and the actual values of θ_2 is about 739600. These three values obtained from experiments consistently match the values derived from the mathematical model by inputting the actual values of four machine-dependent variables. Therefore, our model is deemed valid and can effectively serve as a guide in determining the appropriate values of θ_1, θ_2, and θ_3.

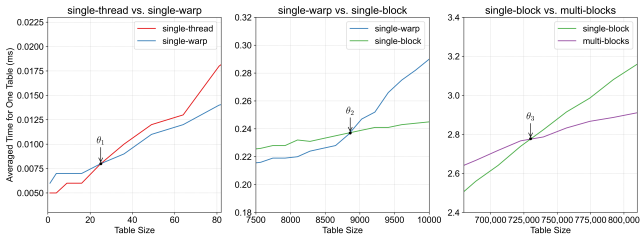

Figure 1: Actual value of thresholds between different approaches on our GPU

For any GPU machine, once the four machine-dependent variables are determined and plugged into this mathematics model, the three thresholds can be calculated. These thresholds serve as criteria to dynamically select different strategies within X-TEDand can be directly used in all types of substantial TED computations running on this machine, which facilitates the efficient utilization of the GPU machine's resources.