1 Terminologies

A *computation graph* is a DAG $G = (V, E)$, whose vertices are the operations performed by different layers in a model in a single timestep, and edges are the tensors that are transferred between different operations. Each operation $v \in V$ has an iteration domain whose size is represented by an ordered set. For e.g., a matrix-matrix multiplication operation (GEMM) that multiplies a matrix of size $m \times k$ with another matrix of size $k \times n$ has a three-dimensional iteration domain of size (m, n, k).

A *parallelization configuration* for an operation is an ordered set of positive integers that describes how the iteration domain of the operation is distributed among p processors. For instance, a configuration $(2, 2, 1)$ for GEMM operation denotes that the iteration domain is split into 2 along m and n dimensions, and distributed among 4 processors.

A *parallelization strategy* for a graph $G = (V, E)$ is a map $\mathcal{S} : V \to (\mathbb{Z}^+)^d$, where $d = \max_{v \in V} v.dim$, that assigns a valid configuration to each vertex in G. For e.g., the strategy $\{(v_1, (2, 1, 1)), (v_2, (1, 2, 1))\}$ denotes that the vertex v_1 is divided among two processors along its first iteration domain dimension, and the vertex v_2 is divided among two processors along its second dimension. An optimal parallelization strategy is the one that minimizes the total execution time of the graph.

Given a strategy $\mathcal{S}$, total execution time of G under $\mathcal{S}$ is given by $t_G(G, \mathcal{S}) = \sum_{v \in V} t_V(v, \mathcal{S}(v)) + \sum_{(u,v) \in E} t_E((u, v), \mathcal{S}(u), \mathcal{S}(v))$, where $t_V(v, c_v)$ provides the time to execute the operation of v (i.e., perform computation and communication such as gradient synchronization, etc.,) under the configuration c_v, and $t_E((u, v), c_u, c_v)$ provides the time to communicate the tensors between operations (e.g., time for all-to-all communication to transpose the matrices).

2 Finding an optimal strategy for a computation graph

A brute-force method to find an optimal strategy is to enumerate all possible strategies, and pick the one that minimizes the total time. However, as this algorithm is exponential in complexity, it is not practical to analyze all possible strategies. Given a strategy $\mathcal{S}$ for G, it can be seen that if we change a configuration $c = \mathcal{S}(v)$ for a vertex v, it only affects the execution time of v and the communication time with its neighbors. We make use of this insight to efficiently compute an optimal strategy in Alg. 1. Particularly, given a vertex v and a strategy $\mathcal{S}$ for G, we can determine what would be the change in total execution time of the graph if we change the configuration for v from, say, c_1 to c_2, by just looking at the configurations of the neighbors of v in $\mathcal{S}$. Hence, if we have a set of possible configurations $\{c_1, \ldots c_n\}$ for v, if changing from c_i to c_j provides a positive difference in execution time, then c_i is sub-optimal under $\mathcal{S}$. This allows us to reduce the search space by removing c_i from consideration for $\mathcal{S}$. A formal proof of optimality is provided in Thm. 1.

Alg. 1 takes a computation graph G and the number of available processors as input, and returns an optimal strategy for G. Line 8 enumerates all possible configurations for a vertex. In lines 12–14 enumerate all possible sub-strategies for a vertex v and its neighbors from their configurations. These sub-strategies provide sufficient information needed to compute the change in the total execution time when we change a configuration for v.

In lines 16–34, in each iteration of the loop, we consider a vertex v in the graph, and we determine what is the best configuration for v (line 25) for each combination of its neighboring configurations. Any sub-optimal configurations are eliminated, thus reducing the search space of the strategies to be validated for optimality.

Once all the vertices have been processed, we collect all remaining valid strategies from the configurations stored in the vertices (after the sub-optimal ones have been eliminated), we choose the one that provides the minimum cost in line 37. In Thm. 1 we show that the final strategy returned by Alg. 1 is an optimal one.

Algorithm 1 Algorithm to determine optimal parallelization strategy for a computation DAG

```
1: procedure FINDOPTSTRATEGY($G = (V, E)$, $nprocs$)
2:     Input(s): $G = (V, E)$ : Computation DAG.
              $(\forall v \in V)[v.dim]$ : Dimension of the iteration domain of the operation of $v$.
              $nprocs$ : Number of processors.
3:     Output(s): $\mathcal{S}$: Optimal parallelization starategy for $G$

4:     ▷ Initialize vertex properties
5:     for all $v \in V$ do
6:         $v.n \leftarrow \{u \mid (u, v) \in E \vee (v, u) \in E\}$                    ▷ Neighbors
7:         $v.un \leftarrow v.n$                                              ▷ Unprocessed neighbors
8:         $v.cfg \leftarrow \{(c_1, c_2, \ldots, c_{v.dim}) \mid (\forall i \in [1, v.dim])[c_i \in \mathbb{Z}^+] \wedge \prod_{i=1}^{v.dim} c_i \leq nprocs\}$
9:                                                        ▷ Set of all possible configurations
10:    end for

11:    ▷ Populate $v.tbl$ with all possible combinations of neighboring configurations. Each entry in $v.tbl$ is
       a sub-strategy for the nodes $\{v\} \cup v.n$
12:    for all $v \in V$ do
13:        $v.tbl \leftarrow \prod_{u \in \{v\} \cup v.n} \{(u, cfg) \mid cfg \in u.cfg\}$
14:    end for

15:    $U \leftarrow V$                                                   ▷ Set of unprocessed vertices
16:    while $U \neq \emptyset$ do
17:        ▷ Choose a vertex with least no. of unprocessed neighbors from $U$
18:        $v_{min} \leftarrow \arg\min_{v \in U} |v.un|$

19:        ▷ Partition $v_{min}.tbl$ such that in each set, all the neighbor configurations are the same. Refer
       Alg. 2 for the function ISVALID
20:        $\mathcal{P} \leftarrow \{S_1, S_2, \ldots \mid \cup_{S_i} = v_{min}.tbl \wedge i \neq j \implies S_i \cap S_j = \emptyset \wedge$
                   $(\forall s_x \in S_i)(\forall s_y \in S_i)[\text{ISVALID}(s_x, s_y, v_{min}.n) = \top] \wedge$
                   $i \neq j \implies (\forall s_x \in S_i)(\forall s_y \in S_j)[\text{ISVALID}(s_x, s_y, v_{min}.n) = \bot]\}$
21:        ▷ Iterate over each set $S \in \mathcal{P}$ and update $v_{min}.tbl$ so that it only contains the sub-strategy in $S$
       that has the least cost.
22:        for all $S \in \mathcal{P}$ do
23:            $best \leftarrow \arg\min_{s \in S} \text{COST}(G, s)$
24:            $rem \leftarrow S \setminus \{best\}$
25:            $v_{min}.tbl \leftarrow v_{min}.tbl \setminus rem$
26:        end for

27:        for all $v \in v_{min}.n$ do
28:            ▷ Restrict the tables of the neighbors based on the best sub-strategies found for $v_{min}$ in Line 37
29:            $common = (v_{min}.n \cap v.n) \cup \{v_{min}, v\}$
30:            $v.tbl \leftarrow \{t_1 \in v.tbl \mid (\exists t_2 \in v_{min}.tbl)[\text{ISVALID}(t_1, t_2, common) = \top])\}$

31:            $v.un \leftarrow v.un \setminus \{v_{min}\}$
32:        end for

33:        $U \leftarrow U \setminus \{v_{min}\}$
34:    end while

35:    ▷ Collect all valid strategies for $G$ from $v.tbl$, and pick the one that minimizes the cost.
36:    $\mathcal{S}' \leftarrow \text{VALIDSTRATEGIES}(G)$                                   ▷ Refer Alg. 3
37:    $\mathcal{S} \leftarrow \arg\min_{S \in \mathcal{S}'} \text{COST}(G, S)$
38:    return $\mathcal{S}$
39: end procedure
```

Algorithm 2 Returns $\top$ if the two sub-strategies S_1 and S_2 have same configurations for the vertices in V.

```
procedure IsValid($S_1, S_2, V$)
    $N_1 \leftarrow \{(n, c) \in S_1 \mid n \in V\}$
    $N_2 \leftarrow \{(n, c) \in S_2 \mid n \in V\}$

    if $N_1 \neq N_2$ then
        return $\perp$
    end if

    return $\top$
end procedure
```

Algorithm 3 Returns a set of valid strategies

```
procedure ValidStrategies($G = (V, E)$)
    $\mathcal{S} \leftarrow \emptyset$
    for all $v \in V$ do
        if $\mathcal{S} = \emptyset$ then
            $\mathcal{S} \leftarrow v.tbl$
            Continue
        end if

        $\mathcal{S}' \leftarrow \emptyset$
        for all $S \in \mathcal{S}$ do
            for all $T \in v.tbl$ do
                $U_1 \leftarrow \{u \mid (u, c) \in S\}$
                $U_2 \leftarrow \{u \mid (u, c) \in v.tbl\}$

                if IsValid($S, T, U_1 \cap U_2$) $= \top$ then
                    $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{S \cup T\}$
                end if
            end for
        end for
        $\mathcal{S} \leftarrow \mathcal{S}'$
    end for

    return $\mathcal{S}$
end procedure
```

3 Proof of optimality

Notation Given a strategy $\mathcal{S}$ of a graph $G = (V, E)$ and $V' \subseteq V$, $\mathcal{S}_{|V'}$ denotes the sub-strategy restricted to the subgraph of G induced by V', and $\mathcal{S}(v)$ denotes the configuration for the vertex v in $\mathcal{S}$. Let $\widehat{\mathcal{S}}$ be an optimal strategy for a computation DAG $G = (V, E)$, and $\widehat{T} = \textsc{Cost}(G, \widehat{\mathcal{S}})$ be the optimal parallelization cost for $\widehat{\mathcal{S}}$.

Alg. 3 takes a DAG G as input, whose vertices have an associated property *tbl* that contains a set of sub-strategies, and returns a set of all possible valid strategies for G. Line 13 of Alg. 1 assigns all possible sub-strategies to each vertex in G. It can be seen that just before entering the loop at line 16 in Alg. 1, we have $\widehat{\mathcal{S}} \in \textsc{ValidStrategies}(G)$.

We will show that the invariant $(\exists \mathcal{S} \in \textsc{ValidStrategies}(G))[\textsc{Cost}(G, \mathcal{S}) \leq \widehat{T}]$ is maintained everywhere after line 14 in Alg. 1.

Theorem 1. *Given an operation DAG $G = (V, E)$ and an optimal parallelization strategy $\widehat{\mathcal{S}}$ for G, the invariant $(\exists \mathcal{S} \in \textsc{ValidStrategies}(G))[\textsc{Cost}(G, \mathcal{S}) \leq \widehat{T}]$ is maintained in any iteration of the loop L in lines 16–34 of Alg. 1.*

Proof. As a contradiction, consider that there exists an iteration l of loop L where the invariant is violated for the first time. Lines 25 and 30 are the only places in loop L where some of the valid sub-strategies are removed from *v.tbl*. We will consider both the cases below.

Let ϕ_l be the set of valid strategies returned by $\textsc{ValidStrategies}(G)$ at the beginning of an iteration l, and ϕ_{l+1} be the set of valid strategies returned at the end of iteration l.

Case I Consider line 25 of Alg. 1. Let $V' = V \setminus \{v_{min}\}$, and $E' = E \cap (V' \times V')$. Note that line 25 only removes some of the configurations of the vertex v_{min}, i.e., $\phi_{l|V'}$ does not change after execution of line 25.

Let B denote the set of all *best* sub-strategies picked in line 25 in iteration l. For some $\mathcal{S} \in B$, let $B(\mathcal{S}_{|v.n})$ denote the configuration c for v s.t. $\{(v, c)\} \cup \mathcal{S}_{|v.n} = \mathcal{S}$. It can be seen that $B(\mathcal{S}_{|v.n})$ is unique from the construction of $\mathcal{P}$ in line 20.

We have,

$$\begin{aligned}
\widehat{T} = {} & \min_{\mathcal{S}\in\phi_l}\{\sum_{v\in V} t_V(v, \mathcal{S}(v)) + \sum_{(u,v)\in E} t_E((u,v), \mathcal{S}(u), \mathcal{S}(v))\} \\
= {} & \min_{\mathcal{S}\in\phi_l}\{t_V(v_{min}, \mathcal{S}(v_{min})) + \sum_{(u,v_{min})\in E} t_E((u, v_{min}), \mathcal{S}(u), \mathcal{S}(v_{min})) + \\
& \sum_{(v_{min},v)\in E} t_E((v_{min}, v), \mathcal{S}(v_{min}), \mathcal{S}(v)) + \sum_{v\in V'} t_V(v, \mathcal{S}(v)) + \sum_{(u,v)\in E'} t_E((u,v), \mathcal{S}(u), \mathcal{S}(v))\} \\
\geq {} & \min_{\mathcal{S}\in\phi_l}\{t_V(v_{min}, B(\mathcal{S}_{|v_{min}.n})) + \sum_{(u,v_{min})\in E} t_E((u, v_{min}), \mathcal{S}(u), B(\mathcal{S}_{|v_{min}.n})) + \\
& \sum_{(v_{min},v)\in E} t_E((v_{min}, v), B(\mathcal{S}_{|v_{min}.n}), \mathcal{S}(v)) + \sum_{v\in V'} t_V(v, \mathcal{S}(v)) + \sum_{(u,v)\in E'} t_E((u,v), \mathcal{S}(u), \mathcal{S}(v))\} \quad (1)
\end{aligned}$$

contradicting our assumption, as there exists a strategy $\mathcal{S} \in \phi_l$ and $\{(v_{min}, B(\mathcal{S}_{|v_{min}.n})) \cup \mathcal{S}_{|V'}\} \in \phi_{l+1}$ whose cost is equal to $\widehat{T}$.

Eq. 1 above follows from the fact that

$$best = \arg\min_{\mathcal{S}_{|V''}} \sum_{v\in\{v_{min}\}\cup v_{min}.n} t_V(v, \mathcal{S}_{|V''}(v)) + \sum_{(u,v)\in E\setminus E'} t_E((u,v), \mathcal{S}_{|V''}(u), \mathcal{S}_{|V''}(v))$$

Case II Consider line 30 in Alg. 1. As the computation in line 30 removes a sub-strategy S from *tbl* only if there doesn't exist any valid strategy that comprises of S, no optimal strategy is eliminated in this step.

□