SUPPLEMENTAL MATERIALS FOR: BEYOND DATA AND MODEL PARALLELISM FOR DEEP NEURAL NETWORKS

Algorithm 1 Full Simulation Algorithm.

Input: An operator graph $\mathcal{G}$, a device topology $\mathcal{D}$, and a parallelization strategy $\mathcal{S}$.
$\mathcal{T}$ = BUILDTASKGRAPH($\mathcal{G}$, $\mathcal{D}$, $\mathcal{S}$)
readyQueue = $\{\}$ *// a priority queue sorted by readyTime*
for t $\in \mathcal{T}_N$ **do**
    t.state = `NOTREADY`
    if $\mathcal{I}(\mathrm{t}) = \{\}$ **then**
        t.state = `READY`
        readyQueue.enqueue(t)
    end if
end for
while readyQueue $\neq \{\}$ **do**
    Task t = readyQueue.dequeue()
    Device d = t.device
    t.state = `COMPLETE`
    t.startTime = $\max\{$t.readyTime, d.last.endTime$\}$
    t.endTime = t.startTime + t.exeTime
    d.last = t
    for n $\in \mathcal{O}(\mathrm{t})$ **do**
        n.readyTime = $\max\{$n.readyTime, t.endTime$\}$
        if all tasks in $\mathcal{I}(\mathrm{n})$ are `COMPLETE` **then**
            n.state = `READY`
            readyQueue.enqueue(n)
        end if
    end for
end while
return $\max\{$t.endTime $\mid$ t $\in \mathcal{T}_N\}$

Algorithm 2 Delta Simulation Algorithm.

1: **Input**: An operator graph $\mathcal{G}$, a device topology $\mathcal{D}$, an original task graph $\mathcal{T}$, and a new configuration c_i' for operator o_i.
2: updateQueue = $\{\}$ *// a priority queue sorted by readyTime*
3: /*UPDATETASKGRAPH *returns the updated task graph and a list of tasks with new* `readyTime`*/
4: $\mathcal{T}$, $\mathcal{L}$ = UPDATETASKGRAPH($\mathcal{T}$, $\mathcal{G}$, $\mathcal{D}$, c_i, c_i')
5: updateQueue.enqueue($\mathcal{L}$)
6: **while** updateQueue $\neq \{\}$ **do**
7:     **Task** t = updateQueue.dequeue()
8:     t.startTime = $\max\{$t.readyTime, t.preTask.endTime$\}$
9:     t.endTime = t.startTime + t.exeTime
10:     **for** n $\in \mathcal{O}$(t) **do**
11:         **if** UPDATETASK(n) **then**
12:             updateQueue.push(n)
13:         **end if**
14:     **end for**
15:     **if** UPDATETASK(t.nextTask) **then**
16:         updateQueue.push(t.nextTask)
17:     **end if**
18: **end while**
19: **return** $\max\{$t.endTime $\mid$ t $\in \mathcal{T}_N\}$
20:
21: **function** UPDATETASK(t)
22:     t.readyTime = $\max\{$p.endTime $\mid$ p $\in \mathcal{I}$(t)$\}$
23:     */*Swap t with other tasks on the device to maintain FIFO.*/*
24:     t.startTime = $\max\{$t.readyTime, t.preTask.endTime$\}$
25:     **if** t's readyTime or startTime is changed **then**
26:         **return True**
27:     **else**
28:         **return False**
29:     **end if**
30: **end function**

1 FULL SIMULATION ALGORITHM

Algorithm 1 shows the pseudocode of the full simulation algorithm. It first builds a task graph using the method described in Section 5.1 and then sets the properties for each task using a variant of Dijkstra's shortest-path algorithm (Cormen et al., 2009). Tasks are enqueued into a global priority queue when ready (i.e., all predecessor tasks are completed) and are dequeued in increasing order by their `readyTime`. Therefore, when a task t is dequeued, all tasks with an earlier `readyTime` have been scheduled, and we can set the properties for task t while maintaining the FIFO scheduling order (assumption A3).

2 DELTA SIMULATION ALGORITHM

Algorithm 2 shows the pseudocode of the full simulation algorithm. It first updates tasks and dependencies from an existing task graph and enqueues all modified tasks into a global priority queue (line 4-5). Similar to the Bellman-Ford shortest-path algorithm (Cormen et al., 2009), the delta simulation algorithm iteratively dequeues updated tasks and propagates the updates to subsequent tasks (line 6-14). The full and delta simulation algorithms always produce the same timeline for a given task graph.

REFERENCES

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.