

1 Recurrence

$$\min_cost(v, SS_u) = \min_{cfg_v} \{ \text{vert_cost}(v, cfg_v) + \sum_{n \in N_u(v)} \text{edge_cost}(v, n, cfg_v, SS_u(n)) + \sum_{n \in N_p(v)} \min_cost(n, SS_u \cup \{cfg_v\}) \}$$

where, $N_u(v)$ is the set of unprocessed neighbors of v , $N_p(v)$ is the set of processed neighbors of v , $\sigma : V^* \rightarrow V^*$ is a function that maps a set of processed vertices to their dependent unprocessed vertices, and SS_u is a map from $\sigma(N_p(v))$ to their configurations.

2 Proof of optimality

3 Algorithm

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

```

1: procedure FINDOPTSTRATEGY( $G = (V, E)$ ,  $nprocs$ )
2:   for all  $v \in V$  do
3:      $v.processed \leftarrow \perp$ 
4:      $v.neigh \leftarrow \{u \in V \mid (u, v) \in E \vee (v, u) \in E\}$ 
5:      $v.depends \leftarrow v.neigh$ 
6:      $v.tbl \leftarrow \{(v, (c_1, c_2, \dots, c_{v.dim})) \mid \forall_{i \in [1, v.dim]} [c_i \in \mathbb{Z}^+] \wedge \prod_{i=1}^{v.dim} c_i \leq nprocs\}$ 
7:      $\triangleright$  Set of all possible configurations
8:   end for

9:    $v \leftarrow \text{GETNEXTVERTEX}(V, \perp)$ 
10:   $opt \leftarrow \emptyset$ 
11:  while  $v \neq \perp$  do
12:     $n_p \leftarrow \{n \in v.neigh \mid n.processed = \top\}$   $\triangleright$  Processed neighbors of  $v$ 
13:     $n_u \leftarrow v.neigh \setminus n_p$   $\triangleright$  Unprocessed neighbors of  $v$ 

14:     $\triangleright$  Generate sub-strategies for  $v$  and its surrounding nodes, by combining configs/sub-strategies of  $v$  and its neighbors. Equivalent to natural join operation on the tables of  $\{v\} \cup v.neigh$ .
15:     $tbl \leftarrow v.tbl$ 
16:    for all  $n \in v.neigh$  do
17:       $tbl \leftarrow \{ss_i \cup ss_j \mid ss_i \in tbl \wedge ss_j \in n.tbl \wedge \text{ISVALID}(ss_i, ss_j)\}$ 
18:    end for
19:     $costs \leftarrow \text{GETCOSTS}(v, n_u, n_p, tbl)$ 

20:     $v.costs \leftarrow \text{GETMINCOSTS}(costs, n_u)$ 
21:     $v.tbl \leftarrow \{cost(1) \mid cost \in v.costs\}$ 
22:     $v.processed \leftarrow \top$ 

23:     $U \leftarrow \{cfg(1) \mid cfg \in v.tbl\}$   $\triangleright U$  is the vertex set in the sub-graph that was processed in this iteration
24:    for all  $u \in U$  do
25:       $u.tbl \leftarrow v.tbl$ 
26:       $u.costs \leftarrow v.costs$ 
27:    end for
28:     $opt \leftarrow v.tbl$ 
29:     $v \leftarrow \text{GETNEXTVERTEX}(V, v)$ 
30:  end while

31:   $\triangleright v.tbl$  of the last vertex processed has exactly a single entry, that corresponds to the optimal strategy for  $G$ .
32:  return  $opt$ 
33: end procedure

```

Algorithm 2 Returns \top if combining two sub-strategies is valid. Two sub-strategies are considered to be invalid if there exists at least one vertex with different configurations in the two sub-strategies.

```

1: procedure ISVALID( $ss_1, ss_2$ )
2:   for all  $cfg_1 \in ss_1$  do
3:     for all  $cfg_2 \in ss_2$  do
4:       if  $cfg_1(1) = cfg_2(1) \wedge cfg_1(2) \neq cfg_2(2)$  then
5:         return  $\perp$ 
6:       end if
7:     end for
8:   end for

9:   return  $\top$ 
10: end procedure

```

Algorithm 3 Computes costs for different sub-strategies.

```

1: procedure GETCOSTS( $v, n_u, n_p, tbl$ )
2:    $costs \leftarrow \emptyset$ 
3:   for all  $ss \in tbl$  do
4:      $cost \leftarrow 0$ 
5:      $ss_p \leftarrow \{cfg \in ss \mid cfg(1) \in n_p\}$   $\triangleright$  Processed neighbors' configs

6:     for all  $cfg \in ss$  do
7:        $u \leftarrow cfg(1)$ 
8:       if  $u = v$  then
9:          $\triangleright$  Add vertex cost for the current vertex  $v$ .
10:         $cost \leftarrow cost + \text{GETVERTEXCOST}(v, cfg)$ 
11:      else if  $u \in n_u$  then
12:         $\triangleright$  Add edge costs for edges between  $v$  and its unprocessed neighbor  $u$ .
13:         $cost \leftarrow cost + \text{GETEDGECOST}(v, u, cfg)$ 
14:      else if  $u \in n_p$  then
15:         $\triangleright$  Add the costs for sub-strategies of processed neighbor  $u$  from its tbl.
16:         $cost_p \leftarrow \exists c_p [c_p \in u.costs \wedge \text{ISVALID}(c_p(1), ss_p)]$ 
17:         $cost \leftarrow cost + cost_p(2)$ 
18:      end if
19:    end for

20:     $costs \leftarrow costs \cup \{(ss, cost)\}$ 
21:  end for
22:  return  $costs$ 
23: end procedure

```

Algorithm 4 Returns \top if configs of n_u are the same in all sub-strategies of $costs(1)$

```

1: procedure HASSAMESS( $costs, n_u$ )
2:   for all  $cost_i \in costs$  do
3:     for all  $cost_j \in costs$  do
4:        $ss_i \leftarrow \{cfg \in cost_i(1) \mid cfg(1) \in n_u\}$ 
5:        $ss_j \leftarrow \{cfg \in cost_j(1) \mid cfg(1) \in n_u\}$ 
6:       if ISVALID( $ss_i, ss_j$ ) =  $\perp$  then
7:         return  $\perp$ 
8:       end if
9:     end for
10:  end for
11:  return  $\top$ 
12: end procedure

```

Algorithm 5 Minimizes $costs$ over processed neighbors.

```

1: procedure GETMINCOSTS( $costs, n_u$ )
2:    $\triangleright$  Partition  $costs$  into mutually disjoint sets, such that all the elements of a set have the same sub-
   strategy for unprocessed vertices  $n_u$ .
3:    $\mathcal{P} \leftarrow \{C_1, C_2, \dots \mid i \neq j \implies C_i \cap C_j = \emptyset \wedge \cup_i C_i = costs \wedge \forall_i [HASSAMESS(C_i, n_u)]\}$ 

4:    $\triangleright$  Extract the strategy with minimum cost from each set of  $\mathcal{P}$ .
5:    $min\_costs \leftarrow \emptyset$ 
6:   for all  $P \in \mathcal{P}$  do
7:      $min\_cost \leftarrow \arg \min_{cost \in P} cost(2)$ 
8:      $min\_costs \leftarrow min\_costs \cup \{min\_cost\}$ 
9:   end for

10:  return  $min\_costs$ 
11: end procedure

```

Algorithm 6 Returns the next vertex to be processed.

```

1: procedure GETNEXTVERTEX( $V, v_{prev}$ )
2:   if  $v_{prev}$  then
3:     for all  $v \in v_{prev}.depends$  do
4:        $v.depends \leftarrow (v.depends \cup v_{prev}.depends) \setminus \{v_{prev}, v\}$ 
5:     end for
6:   end if

7:    $V_u \leftarrow \{v \in V \mid v.processed = \perp\}$ 
8:   if  $V_u = \emptyset$  then
9:     return  $\perp$ 
10:  end if

11:   $v_{next} \leftarrow \arg \min_{v \in V_u} |v.depends|$ 
12:  return  $v_{next}$ 
13: end procedure

```
