1 Recurrence

$$min\_cost(v, SS_u) = \min_{cfg_v} \{vert\_cost(v, cfg_v) + \sum_{n \in N_u(v)} edge\_cost(v, n, cfg_v, SS_u(n)) + \sum_{n \in N_p(v)} min\_cost(n, SS_u \cup \{cfg_v\})\}$$

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

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

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

        $\triangleright$ 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 $\{v\} \cup v.neigh$.
        $tbl \leftarrow v.tbl$
        for all $n \in v.neigh$ **do**
            $tbl \leftarrow \{ss_i \cup ss_j \mid ss_i \in tbl \wedge ss_j \in n.tbl \wedge$ ISVALID$(ss_i, ss_j)\}$
        end for
        $costs \leftarrow$ GETCOSTS(v, n_u, n_p, tbl)

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

        $U \leftarrow \{cfg(1) \mid cfg \in v.tbl\}$ $\triangleright$ U is the vertex set in the sub-graph that was processed in this iteration
        for all $u \in U$ **do**
            $u.tbl \leftarrow v.tbl$
            $u.costs \leftarrow v.costs$
        end for
        $opt \leftarrow v.tbl$
        $v \leftarrow$ GETNEXTVERTEX(V, v)
    end while

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

procedure $\textsc{IsValid}(ss_1, ss_2)$
    for all $cfg_1 \in ss_1$ **do**
        for all $cfg_2 \in ss_2$ **do**
            if $cfg_1(1) = cfg_2(1) \wedge cfg_1(2) \neq cfg_2(2)$ **then**
                return $\bot$
            end if
        end for
    end for

    return $\top$
end procedure

Algorithm 3 Computes costs for different sub-strategies.

procedure $\textsc{GetCosts}(v, n_u, n_p, tbl)$
    $costs \leftarrow \emptyset$
    for all $ss \in tbl$ **do**
        $cost \leftarrow 0$
        $ss_p \leftarrow \{cfg \in ss \mid cfg(1) \in n_p\}$ ▷ Processed neighbors' configs

        for all $cfg \in ss$ **do**
            $u \leftarrow cfg(1)$
            if $u = v$ **then**
                ▷ Add vertex cost for the current vertex v.
                $cost \leftarrow cost + \textsc{GetVertexCost}(v, cfg)$
            else if $u \in n_u$ **then**
                ▷ Add edge costs for edges between v and its unprocessed neighbor u.
                $cost \leftarrow cost + \textsc{GetEdgeCost}(v, u, cfg)$
            else if $u \in n_p$ **then**
                ▷ Add the costs for sub-strategies of processed neighbor u from its `tbl`.
                $cost_p \leftarrow \exists c_p[c_p \in u.costs \wedge \textsc{IsValid}(c_p(1), ss_p)]$
                $cost \leftarrow cost + cost_p(2)$
            end if
        end for

        $costs \leftarrow costs \cup \{(ss, cost)\}$
    end for
    return $costs$
end procedure

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

procedure HasSameSS($costs, n_u$)
  for all $cost_i \in costs$ **do**
    for all $cost_j \in costs$ **do**
      $ss_i \leftarrow \{cfg \in cost_i(1) \mid cfg(1) \in n_u\}$
      $ss_j \leftarrow \{cfg \in cost_j(1) \mid cfg(1) \in n_u\}$
      if IsValid(ss_i, ss_j) $= \bot$ **then**
        return $\bot$
      end if
    end for
  end for
  return $\top$
end procedure

Algorithm 5 Minimizes *costs* over processed neighbors.

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

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

  return min_costs
end procedure

Algorithm 6 Returns the next vertex to be processed.

procedure GetNextVertex(V, v_{prev})
  if v_{prev} **then**
    for all $v \in v_{prev}.depends$ **do**
      $v.depends \leftarrow (v.depends \cup v_{prev}.depends) \setminus \{v_{prev}, v\}$
    end for
  end if

  $V_u \leftarrow \{v \in V \mid v.processed = \bot\}$
  if $V_u = \emptyset$ **then**
    return $\bot$
  end if

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