System Model

A cluster is characterized by a set P = {p_1, p_2,..., p_m} of processors connected by high speed interconnect. It is assumed that the computational nodes are homogeneous in nature, meaning that all processors are identical in their capabilities. Similarly, the underlying interconnection is assumed to be homogeneous and, thus, communication overhead of a message with fixed data size between any pair of nodes is considered to be the same. Each node communicates with other nodes through message passing, and the communication time between two precedence constrained tasks assigned to the same node is negligible. Parallel applications with a set of precedence-constrained tasks can be represented in form of a Directed Acyclic Graph (DAG). a parallel application running in clusters is modeled as a vector (V, E), where V = {v_1, v_2, ..., v_n} represents a set of precedence constrained parallel tasks, and E denotes a set of messages representing communications and precedence constraints among parallel tasks. The weight on the vertex represents the execution time of the task and the weight on the edge represents the communication cost between two tasks. For each task in V, t_i is defined as the required time to compute v_i. Similarly, $e_{ij} = (v_i, v_j) \in E$ is defined as a message transmitted from task v_i to v_j, and c_{ij} is the required time of passing the message $e_{ij} \in E$. Please note that e_{ij} is set to zero if v,and v_j are assigned to the same processor.

Parameters Calculation and Scheduling Decisions Algorithm

Phase 1: Generate Original Task Scheduling Sequence

Precedence constraints of a set of parallel tasks have to be guaranteed by executing predecessor tasks before successor tasks. To achieve this goal, the first step in your algorithm is to generate an ordered task sequence using the concept of level. The level of each task is defined as the computation time from current task to the exit task. There are alternative ways to generate the task sequence for a DAG, we calculate the level $L(v_i)$ of task v_i using equation 1:

$$L(v_i) = \begin{cases} t_i, & \text{if successor(i)} = \Phi \\ \max\underbrace{\left(level(k)\right)}_{k \in succ(i)} + t_i & otherwise \end{cases} \quad (1)$$

The levels of other tasks can be calculated in a bottom-up fashion by recursively applying the second term on the right-hand side of Eq. (1). Once we obtain the levels, tasks will be sorted in ascending order of the levels and the sorted tasks form the original task scheduling sequence.

Phase 2: Parameters Calculation

The second phase in your algorithm is to calculate important parameters, which your algorithms rely on to make scheduling decision. The important notation and parameters are listed in Table 1.

The earliest start time of the entry task is 0 (see the first term on the right side of Eq. (2). The earliest start times of all the other tasks can be calculated in a top-down manner by recursively applying the second term on the right side of Eq. (2).

TABLE 1 Important Notations and Parameters

Notation	Definition
$EST(v_i)$	Earliest start time of task v_i
$ECT(v_i)$	Earliest completion time of task v_i
$FP(v_i)$	Favorite predecessor of task v_i
$LACT(v_i)$	Latest allowable completion time of task v_i
$LAST(v_i)$	Latest allowable start time of task v_i

$$EST(v_i) = \begin{cases} 0, & \text{if predecessor(i)} = \Phi, \\ \min\left(\max_{e_{ki} \in E, v_k \neq v_j} \left(ECT(v_j), ECT(v_k) + c_{ki} \right) \right), & \text{otherwise} \end{cases} \quad (2)$$

The earliest completion time of task v_i is expressed as the summation of its earliest start time and execution time. Thus, we have

$$ECT(v_i) = EST(v_i) + t_i. \quad (3)$$

Allocating task v_i and its favorite predecessor $FP(v_i)$ on the same processor can lead to a shorter schedule length. As such, the favorite predecessor $FP(v_i)$ is defined as below

$$FP(v_i) = v_j, \text{where } \forall e_{ji} \in E, e_{ki} \in E, j \neq k \left| ECT(v_j) + c_{ji} \geq ECT(v_k) + c_{ki}. \right. \quad (4)$$

As shown by the first term on the right-hand side of Eq. (5), the latest allowable completion time of the exit task equals to its earliest completion time. The latest allowable completion times of all the other tasks are calculated in a bottom-up manner by recursively applying the second term on the right-hand side of Eq. (5).

$$LACT(v_i) = \begin{cases} ECT(v_i), & \text{if successor(i)} = \Phi, \\ \min\left(\min_{e_{ij} \in E, v_i \neq FP(v_j)} \left(LAST(v_j) - c_{ij} \right), \min_{e_{ij} \in E, v_i = FP(v_j)} \left(LAST(v_j) \right) \right), & \text{otherwise} \end{cases} \quad (5)$$

The latest allowable start time of task v_i is derived from its latest allowable completion time and execution time. Hence, the $LAST(v_i)$ can be written as

$$LAST(v_i) = LACT(v_i) - t_i. \quad (6)$$

Phase 3: Scheduling Decision

The final scheduling decision will be made as follows:

1. All tasks in the same critical path will be allocated to the same processor in order to reduce communication cost.
2. Duplicate tasks when the schedule length can be reduced.

```
1.  v = first waiting task of scheduling queue;
2.  i = 1;
3.  assign v to P_i;
4.  while (not all tasks are allocated to computational nodes) do
5.          u = FP(v);
6.          if (u has already been assigned to another processor) then
7.                  if (LAST(v) - LACT(u)<c_uv) then /* if duplicate u, we can shorten the schedule length */
8.                          assign u to P_i; /*duplicate u*/
9.          else allocate u to P_i;
10.         v = u;
11.         if v is entry task then
12.                 v = the next task that has not yet been allocated to a computational node;
13.                 i++;
14.                 assign v to P_i;
```