Program Parallelization for Effective Use of Computational Resources

計算資源の有効利用を目的とした プログラム並列化

February 2012

Information Network System II,

Global Information and Telecommunication Studies,

Graduate School of Global Information and

Telecommunication Studies, Waseda University

Hidehiro KANEMITSU

Contents

List of Figures

List of Tables

List of Notations

n_k^s	The k-th task after s task merging steps.
$e_{i,j}^s$	The precedence relationship from n_i^s to n_j^s.
$w(n_k^s)$	Size of n_k^s, i.e., time units required to be processed by the reference PE.
$c(e_{i,j}^s)$	Size of data by $e_{i,j}^s$, i.e., time units required to be transferred from n_i^s to n_j^s on the reference communication link.
V_s	The set of tasks after s task merging steps.
E_s	The set of edges (precedence relationships) after s task merging steps.
V_{cls}^s	The set of clusters after s task merging steps.
$G_{cls}^s = (V_s, E_s, V_{cls}^s)$	A task graph expressed as a DAG.
$top_s(i)$	$\{n_k^s \mid \forall n_l^s \in pred(n_k^s) s.t., n_l^s \notin cls_s(i)\} \cup$ START Tasks $\in cls_s(i)$.
$in_s(i)$	$\{n_k^s \mid \exists n_l^s \in pred(n_k^s) s.t., n_l^s \notin cls_s(i)\} \cup$ START Tasks $\in cls_s(i)$.
$out_s(i)$	$\{n_k^s \mid \exists n_l^s \in suc(n_k^s) s.t., n_l^s \notin cls_s(i)\} \cup$ END Tasks $\in cls_s(i)$.
$btm_s(i)$	$\{n_k^s \mid \forall n_l^s \in suc(n_k^s), s.t., n_l^s \notin cls_s(i)\} \cup$ END Tasks $\in cls_s(i)$.
$desc(n_k^s, i)$	$\{n_l^s \mid n_k^s \prec n_l^s, n_l^s \in cls_s(i)\} \cup \{n_k^s\}$.
$S(n_k^s, i)$	$\sum_{n_l^s \in cls_s(i)} w(n_l^s) - \sum_{n_l^s \in desc(n_k^s, i)} w(n_l^s)$.
$tlevel(n_k^s)$	$\begin{cases} \max\limits_{n_l^s \in pred(n_k^s)} \{tlevel(n_l^s) + w(n_l^s) + c(e_{l,k})\}, \ where\ n_k^s \in top_s(i), \\ TL_s(i) + S(n_k^s, i), \ otherwise. \end{cases}$
$TL_s(i)$	$\max\limits_{n_k^s \in top_s(i)} \{tlevel(n_k^s)\}$.
$blevel(n_k^s)$	$\max\limits_{n_l^s \in suc(n_k^s)} \left\{ w(n_k^s) + c(e_{k,l}^s) + blevel(n_l^s) \right\}$.
$level(n_k^s)$	$tlevel(n_k^s) + blevel(n_k^s)$.
$BL_s(i)$	$\max\limits_{n_k^s \in out_s(i)} \{S(n_k^s, i) + blevel(n_k^s)\}$.
$LV_s(i)$	$TL_s(i) + BL_s(i) = \max\limits_{n_k^s \in cls_s(i)} \{level(n_k^s)\}$.
$sl_w(G_{cls}^s)$	WSL (Worst Schedule Length), i.e., $\max\limits_{cls_s(i) \in V_{cls}^s} \{LV_s(i)\}$.
seq_s	Set of tasks by which $sl_w(G_{cls}^s)$ is derived.
p	One path of G_{cls}^0, i.e., $\{n_0^0, n_1^0, n_2^0, \ldots, n_k^0\} \cup \{e_{0,1}^0, e_{1,2}^0, \ldots e_{k-1,k}^0\}$, by which a sequence $< n_0^0, n_1^0, n_2^0, \ldots, n_k^0 >$ is constructed, where $e_{l-1,l}^0 \in E_0$. n_0^0 is a START task and n_k^0 is an END task.

p'	One subpath of G^0_{cls}, i.e., $\{n^0_0, n^0_1, n^0_2, \ldots, n^0_k\} \cup \{e^0_{0,1}, e^0_{1,2}, \ldots e^0_{k-1,k}\}$, by which a sequence $< n^0_0, n^0_1, n^0_2, \ldots, n^0_k >$ is constructed, where $e^0_{l-1,l} \in E_0$. n^0_0 is not a START task or n^0_k is not an END task.
$seq^{\prec}_s$	One path in which every task belongs to seq_s.
$seq^{\prec}_s(i)$	Set of subpaths in each of which every task in $cls(i)$ belongs to $seq^{\prec}_s$.
w_{max}	$\max\limits_{cls_R(i) \in V^R_{cls}} \{w(cls_R(i))\} - \delta$.
ϕ	$\sharp$ of clusters in which at least one task belongs to $seq^{\prec}_s$.
cp	Critical path length of G^0_{cls}.
cp_w	$\max\limits_{p \in G^0_{cls}} \left\{ \sum\limits_{n^0_k \in p} w(n^0_k) \right\}$.
$g_{\min}$	$\min\limits_{n^0_k \in V^0_{cls}} \left\{ \dfrac{\min\limits_{n^0_j \in pred(n^0_k)} \{w(n^0_j)\}}{\max\limits_{n^0_j \in pred(n^0_k)} \{c(e^0_{j,k})\}}, \dfrac{\min\limits_{n^0_l \in suc(n^0_k)} \{w(n^0_l)\}}{\max\limits_{n^0_l \in suc(n^0_k)} \{c(e^0_{k,l})\}} \right\}$.
$g_{\max}(n^s_k)$	$\max \left\{ \dfrac{\max\limits_{n^0_j \in pred(n^0_k)} \{w(n^0_j)\}}{\min\limits_{n^0_j \in pred(n^0_k)} \{c(e^0_{j,k})\}}, \dfrac{\max\limits_{n^0_l \in suc(n^0_k)} \{w(n^0_l)\}}{\min\limits_{n^0_l \in suc(n^0_k)} \{c(e^0_{k,l})\}} \right\}$.
Δsl_w	$sl_w(G^R_{cls}) - sl_w(G_{org}) = sl_w(G^R_{cls}) - cp$.
$\Delta sl_{w,up}$	An upper bound of Δsl_w.
$len(seq^{\prec}_s(i))$	$\sum\limits_{n^s_k \in seq^{\prec}_s(i)} w(n^s_k) + \sum\limits_{\substack{n^s_k, n^s_l \in seq^{\prec}_s(i), \\ e^0_{k,l} \in E_0}} c(e^0_{k,l})$.
$len(seq^{\prec}_s)$	$\sum\limits_{n^s_k \in seq^{\prec}_s} w(n^s_k) + \sum\limits_{\substack{n^s_k, n^s_l \in seq^{\prec}_s, \\ e^0_{k,l} \in E_0}} c(e^0_{k,l})$.
ΔL_i	$\sum_{n^R_k \in cls_R(i) \cap seq_R} w(n^R_k) - len(seq^{\prec}_R(i))$.
ΔL_w	$sl_w(G^R_{cls}) - len(seq^{\prec}_R)$.
$\Delta L_{w,up}$	An upper bound of ΔL_w.
P_i	The i-th PE.
P	$\{P_1, P_2, \ldots, P_m\}$.
α_i	The processing speed of P_i.
α	$\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$.
$\beta_{i,j}$	The communication bandwidth from P_i to P_j.
β	$\{\beta_{1,2}, \ldots \beta_{i,j}, \ldots, \}$.
ϕ_s	$\{\ldots, < cls_s(i), P_p >, \ldots\}$.
$sl_w(G^s_{cls}, \phi_s)$	WSL (Worst Schedule Length) in the mapping ϕ_s, i.e., $\max\limits_{cls_s(i) \in V^s_{cls}} \{LV_s(i)\}$.
$len(p, \phi_0)$	$\sum\limits_{n^0_k \in p} t_p(n^0_k, \max\limits_{\alpha_i \in \alpha} \{\alpha_i\}) + \sum\limits_{e^0_{k,l} \in p} t_c(c(e^0_{k,l})), \max\limits_{\beta_{i,j} \in \beta} \{\beta_{i,j}\})$.
$cp(\phi_s)$	$\max\limits_{p} \left\{ \sum\limits_{n^s_k \in p} t_p(n^s_k, \alpha_p) + \sum\limits_{e^s_{k,l} \in p} t_c(c(e^s_{k,l}), \beta_{p,q}) \right\}$, where n^s_k, n^s_l are assigned to P_p, P_q, respectively.

$cp_w(\phi_s)$ $\max\limits_{p \in G^s_{cls}} \left\{ \sum\limits_{n^s_k \in p} t_p(n^s_k, \alpha_p) \right\}$.

$g_{\min}(\phi_s)$ $\min\limits_{n^s_k \in V^s_{cls}} \left\{ \dfrac{\min\limits_{n^s_j \in pred(n^s_k)} \left\{ t_p(n^s_j \alpha_p) \right\}}{\max\limits_{n^s_j \in pred(n^s_k)} \left\{ t_c(e^s_{j,k}, \beta_{p,q}) \right\}}, \dfrac{\min\limits_{n^s_l \in suc(n^s_k)} \left\{ t_p(n^s_l, \alpha_r) \right\}}{\max\limits_{n^s_l \in suc(n^s_k)} \left\{ t_c(e^s_{k,l}, \beta_{q,r}) \right\}} \right\}$.

$\phi_{s,identical}$ A mapping of clusters to identical processors after s task merging steps.

Chapter 1

Introduction

1.1 Background

Computer technologies, i.e., how to accelerate the processing power, how to expand the capability, and how to realize an application, have been developed on both hardware and software aspects. High performance computing models, such as high throughput computing [5] and grid computing [6, 8, 9] have been taking a critical role in such developments. One of objectives of those computing models is typically to minimize the response time. On the other hand, a high throughput computing [5] focuses on maximizing computing throughput (the amount of computation per a time unit). This means that an objective of a high performance computing model depends on the situation and the environment to be assumed.

In the light of realization for a high performance computing model, a number of programming models and middlewares exist. In the case of programming models, a programming standard (MPI [13]) and API sets (PVM [14], MPICH [15]) are applied for realizing interprocessor communications. By using such programming models and middlewares, the program can be transformed into the one which can be executed in parallel or concurrently among processors in the computer and/or over the network. As for the common terminologies, the program submitted into the system is defined as "a job," and each execution unit consisting the job is defined as "task." How to schedule each task is generally known as task scheduling problem [4]. Though whether the optimal schedule can be obtained or not depends on the task execution model, at least how to schedule the job composed of the set of tasks is known as a NP-complete problem [4]. Many researches have tried to find a near-optimal solution within the practical running time.

As for a real situation, one of major trends in task execution models is to divide the required data into several pieces and then distribute them to workers such as "a master - worker model." In contrast to such a data intensive job, how to divide a computation intensive job into several execution units for parallel execution is under discussion from theoretical points of view. If we take task parallelization into account in a grid environment such as a computational grid [7, 10, 11], a task scheduling strategy should be established to achieve effective use of processors, which means to maximize the degree of contribution per a processor to the reduction of the response time. However, every conventional approach

has no criterion to achieve the goal. For example, one method to decide the set of processors is to merge several tasks into one assignment unit [17, 18, 24, 26, 27, 29, 32, 42–44]. In such a method, each assignment unit is generated according to the specific criterion. Hence, both the response time and the number of processors obtained depends on the criterion. The problem in conventional approaches to achieve effective use of processors is that the number of generated assignment unit can be too many because such approaches try to minimize the response time only.

1.2 Research target

The objective of the dissertation is to propose the theoretical method for achieving effective use of processors in distributed systems, where each processor is connected to others over the network. In a real situation, the system to be assumed is roughly classified into two types, i.e., a homogeneous distributed system and heterogeneous distributed systems. In the former, both each processing speed and each communication bandwidth are identical, while in the latter their values are arbitrary. Thus, the dissertation mainly consists of two parts in terms of the system to be assumed, i.e., how to achieve effective use of processors in homogeneous distributed systems (chapter 3) or heterogeneous distributed systems (chapter 4). The basic concept behind the research is to impose the lower bound to each assignment unit (cluster) size for effective use of processors, thereby the number of processors is limited to some extent. Thus, in both two parts, the main issue to be solved is how to theoretically derive the lower bound.

1.3 Abstract of each chapter

Figure 1.1 shows relationships among chapters. Before describing those two parts, the job model is defined (chapter 2). The job is the one which is abstracted from a program, in which each statement and function call are handled as a task. On the other hand, each data exchanged among tasks corresponds to a communication over the network. If we assume a general purpose job, which can be modeled as a DAG (Directed Acyclic Graph), where a task corresponds to a node and a data corresponds to an edge. The specific model we assume is how to parallelize the DAG type job over the completely connected network for effective use of processors. In the first part (chapter 3), i.e., in homogeneous distributed systems model, the lower bound for every asignment unit (cluster) is statically derived before each task is scheduled. In other words, we have no way to decide the response time at the derivation phase. Hence, it is necessary to derive the lower bound with estimating the response time. The assumed procedure is to estimate the response time using the indicative value, while each cluster is generated by a task merging step. We define the indicative value as WSL (Worst Schedule Length, denoted as $sl_w(G^s_{cls})$) which means the largest value the response time can take when every task is executed as late as possible after s task merging steps. G^s_{cls} means the state that the set of clusters and communications after s task merging steps have been performed. Since the variation of indicative value must have effect on the minimization of the response time, we theoretically analyze the relationship between $sl_w(G^s_{cls})$ and the response time decided after a task scheduling. As a result, two theorems

are proved, i.e., the one is that the reduction of $sl_w(G^s_{cls})$ can lead to the reduction of the lower bound of the response time, and the other is that the reduction of $sl_w(G^s_{cls})$ can lead to the reduction of the upper bound of the response time. Then we estimate the task combination in an cluster when $sl_w(G^R_{cls})$ is effectively reduced (where R is the number of task merging steps when every cluster size exceeds the lower bound δ). By considering δ as a variable, we derive the value of δ as δ_{opt} when $sl_w(G^R_{cls})$ can be minimized. The processes to generate each cluster are as follows.

1. Derive the lower bound δ_{opt} by assuming the ideal structure for each cluster after R task merging steps.
2. Generate each cluster until every cluster size exceeds δ_{opt} with minimizing $sl_w(G^R_{cls})$.

As the second process, we propose the task clustering algorithm (the set of task merging steps) which has the low time complexity and is capable of effectively reducing $sl_w(G^R_{cls})$. Then we prove the following points by experimental simulations.

1. The generated clusters by the proposal have the task combination for each of them for minimizing $sl_w(G^R_{cls})$.
2. The response time can be effectively reduced with the reduction of $sl_w(G^R_{cls})$ by the proposal.
3. The derived lower bound, i.e., δ_{opt} has a good impact on the reduction of the response time.
4. The algorithm running time is practical.
5. Applicability of the proposal in realistic jobs such as a Gaussian Elimination DAG and a FFT DAG.

From results shown by the experiments, we make conclusion that the proposal can achieve effective use of processors in homogeneous distributed systems.

As the second part (chapter 4), we propose how to achieve effective use of processors in heterogeneous distributed systems. Specifically, we present three points as follows.

(1) The lower bound of a cluster execution time (sum of each task execution time in the cluster on a processor) is derived with taking the processor's capability into account.

(2) The policy for selecting the processor to be assigned.

(3) A task clustering algorithm to generate the cluster which is assigned to the processor selected in (2). As a result, the generated cluster execution time exceeds the lower bound derived in (1).

In contrast to the case of homogeneous distributed system, each task execution time depends on not only each processing speed, but also each communication bandwidth. Thus, the lower bound of each cluster execution time should be decided according to the processor which has been selected as the assignment target. At (1), at first we define the indicative value for the response time as $sl_w(G^s_{cls}, \phi_s)$, where ϕ_s means the mapping state between

each cluster and each processor after s task merging steps. Then we prove that the reduction of $sl_w(G^s_{cls}, \phi_s)$ can lead to the reduction of the lower bound of the response time. Also we prove that so is true for the upper bound of the response time. Hence, the fundamental objective of the proposal is to minimize $sl_w(G^R_{cls}, \phi_R)$ by imposing the lower bound for each cluster execution time decided according to each processor and the processor assignment. Then we derive the lower bound of the cluster execution time for the selected processor as the next assignment target. The lower bound is expressed as $\delta^s_{opt}(P_p)$, where P_p is selected before s-th task merging step, and then we estimate the lower bound for every cluster execution time on the path dominating $sl_w(G^{s-1}_{cls}, \phi_{s-1})$, i.e., by which $sl_w(G^{s-1}_{cls}, \phi_{s-1})$ is decided. That is, we derive the lower bound by temporally assuming the homogeneous distributed system by the set of P_p . At the same time, the upper bound of $sl_w(G^s_{cls}, \phi_s)$ is derived as $\Delta sl^{s-1}_{w,up}$. Since $\delta^s_{opt}(P_p)$ is a function of P_p , both the processing speed and the communication bandwidth are variables. Furthermore, $\Delta sl^{s-1}_{w,up}$ is a function of the lower bound for a cluster execution time. Hence, the actual processor to be assigned after s - th task merging step is decided by assigning every processing speed and communication bandwidth in the set of unassigned processors to $\Delta sl^{s-1}_{w,up}$. The processor by which $\Delta sl^{s-1}_{w,up}$ is minimized is selected as the next assignment target. Each cluster is generated by the task clustering algorithm for each derived lower bound. Experimental comparisons are conducted to confirm, (i) the reduction of $sl_w(G^s_{cls}, \phi_s)$ can lead to the reduction of the response time, (ii) optimality of $\delta^s_{opt}(P_p)$, and (iii) optimality of the processor assignment. As a conclusion, the contribution of the dissertation is to propose a new theoretical approach for effective use of processors in distributed systems. The main achievement is to decide how large each assignment unit size should be set. As a future work, more realistic aspects such as communication bottlenecks by hops should be taken into consideration.

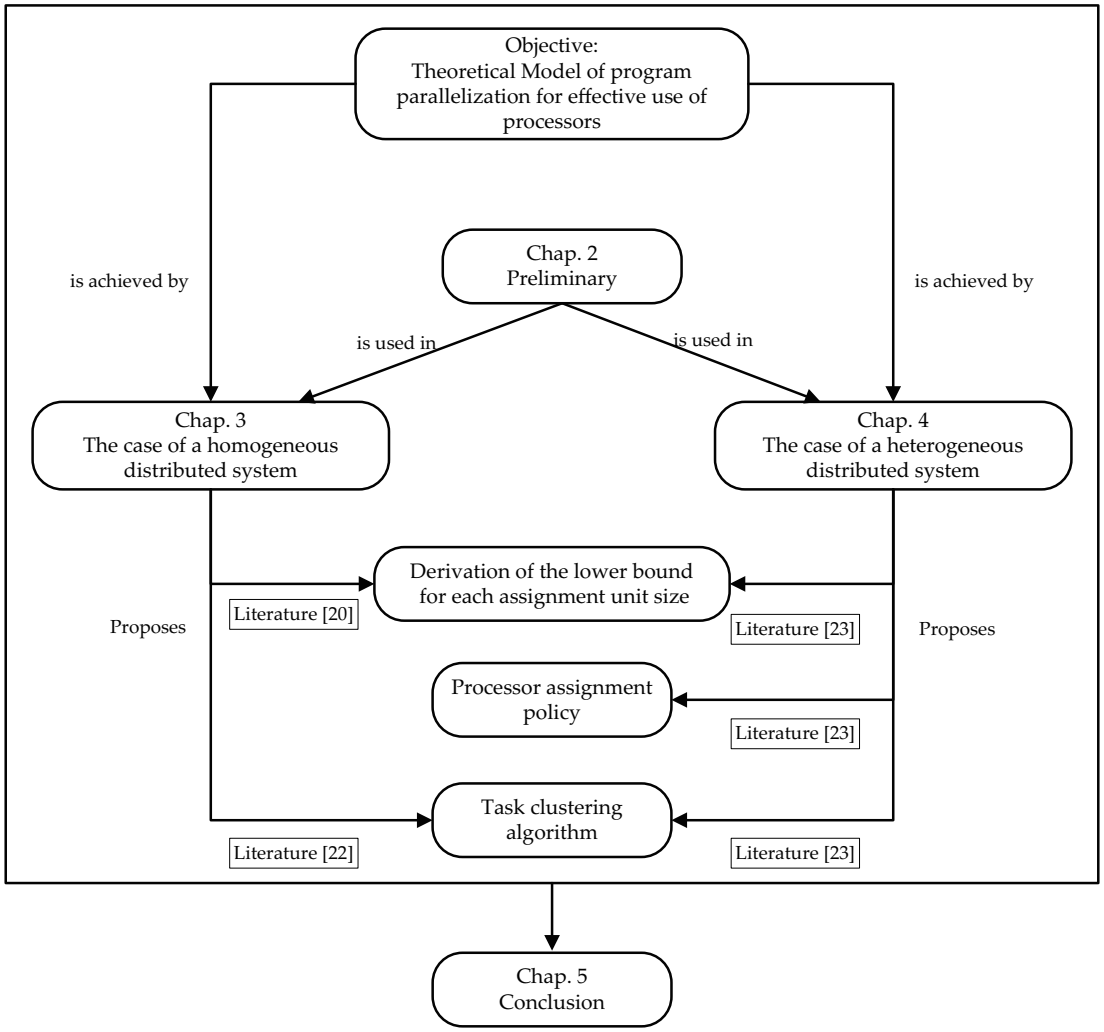


Figure 1.1: Relationships among chapters

Chapter 2

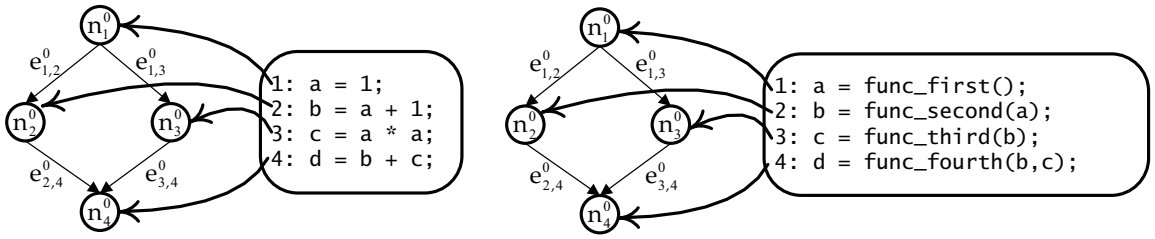
Preliminary

2.1 Job model

We assume a job to be executed among distributed processor elements (PEs) can be expressed as a Directed Acyclic Graph (DAG), which is one of task graphs. At first, let $G^s_{cls} = (V_s, E_s, V^s_{cls})$ be the DAG after s task merging steps. Merging steps will be discussed in the later part of the dissertation. V_s is the set of tasks after s task merging steps, E_s is the set of edges (data communications among tasks) after s task merging steps, and V^s_{cls} is the set of clusters which consists of one or more tasks after s task merging steps. An i-th task is denoted as n^s_i. Let $w(n^s_i)$ be the task size of n^s_i, i.e., the time a reference processor takes to execute the task. A reference processor is the one selected for measuring the processing time of a task, while the reference communication link is the one selected for measuring the data transfer time of a data. Thus, the processing time for each task and the data transfer time for each data are derived by taking ratios in terms of the processing speed and the communication bandwitdh. We denote data dependency and direction of data transfer from n^s_i to n^s_j with $e^s_{i,j}$. And $c(e^s_{i,j})$ is used to denote the data size, i.e., the time to transfer the data from n^s_i to n^s_j over the reference communication link.

One constraint imposed by a DAG is that a task can not be started execution until all data from its predecessor tasks arrive. For instance, $e^s_{i,j}$ means that n^s_j can not be started until data from n^s_i arrives at the PE which will execute n^s_j. And let $pred(n^s_i)$ be the set of immediate predecessors of n^s_i, and $suc(n^s_i)$ be the set of immediate successors of n^s_i. If $pred(n^s_i) = \emptyset$, n^s_i is called START task, and if $suc(n^s_i) = \emptyset$, n^s_i is called END task. If there are one or more paths from n^s_i to n^s_j, we denote such a relation as $n^s_i \prec n^s_j$.

Figure 2.1 shows a DAG example. In (a), it is assumed that each task corresponds to one statement. On the other hand, in (b), each task is assumed to be one function call. In general, a DAG, i.e., a task graph can be generated from a dominance tree [3], which represents data dependencies and control dependencies among tasks. If we assume that one task corresponds to one statement (e.g., only a variable assignment), the total number of tasks in the DAG becomes very large, so that both the total communication overheads and the time taken for a task scheduling become large. Thus, typically many program parallelization compilers adopt the transformation from small granularity to larger granularity in terms of the task structure. For example, in HTG (Hierarchical Task Graph [1, 2]), each



(a) 1 task :1 statement

(b) 1 task :1 function call

Figure 2.1: Example of a DAG.

task corresponds to one execution block, e.g., one looping or one function call in order to reduce both the number of tasks and communication overheads. Thus, in this dissertation we assume that each task corresponds to one function call (i.e., (b) at Figure 2.1).

2.2 Task clustering

The i-th cluster in V_{cls}^s is denoted as $cls_s(i)$. If n_k^s is included in $cls_s(i)$ by "the s-th task merging step," we formulate one task merging step as $cls_{s+1}(i) \leftarrow cls_s(i) \cup \{n_k^s\}$. If any two tasks, i.e., n_i^s and n_j^s, are included in the same cluster, they are assigned to the same PE. Then the communication between n_i^s and n_j^s is localized, so that we define $c(e_{i,j}^s)$ becomes zero. Task clustering is a set of task merging steps, that is finished when a certain criteria has been satisfied. Let one task merging step for $cls_s(i)$ and $cls_s(k)$ be defined as $merge(cls_s(i), cls_s(k))$. This procedure is expressed as

$$
\begin{aligned}
&cls_{s+1}(i) \leftarrow cls_s(i) \cup cls_s(k);\\
&V_{cls}^{s+1} \leftarrow V_{cls}^s - \{cls_s(k)\}, E_{s+1} \leftarrow E_s;\\
&c(e_{p,q}^{s+1}) \leftarrow 0 \; for \; \forall n_p^{s+1}, n_q^{s+1} \in cls_{s+1}(i), e_{p,q}^{s+1} \in E_{s+1};\\
&return \; cls_{s+1}(i);
\end{aligned}
\tag{2.1}
$$

Then one cluster is one assignment unit for a PE.

Let the input DAG for task clustering be $G_{cls}^s = (V_s, E_s, V_{cls}^s)$, where s is the number of task merging steps. Let the set of tasks in V_s be $\{n_1^s, n_2^s, \dots\}$, and let the set of edges in E_s be $\{\dots, e_{k,l}^s, \dots\}$. This means that G_{cls}^s is the DAG just after s task merging steps have been performed to G_0. At the initial state, let $s = 0$ and let $V_0 \leftarrow V, E_0 \leftarrow E$. Furthermore, let $cls_0(1) = \{n_1^0\}, cls_0(2) = \{n_2^0\}, \dots, V_{cls}^0 \leftarrow \{cls_0(1), cls_0(2), \dots\}$. The sum of each task size in $cls_s(i)$ is defined as $w(cls_s(i))$, and in this dissertation $w(cls_s(i))$ is called "cluster size" of $cls_s(i)$.

Throughout this dissertation, we denote that $cls_s(i)$ is "linear" if and only if $cls_s(i)$ contains no independent task [24], i.e., every task in $cls_s(i)$, has precedence relationships with other tasks. Note that if one cluster is linear, at least one path among any two tasks in the cluster exists and the task execution order is unique.

2.3 System model

We have two assumptions in terms of the system in this dissertation, i.e., homogeneous distributed systems described in chapter 3 and heterogeneous distributed systems described in chapter 4.

In both chapters, every PE can independently execute a task and send/receive data at the same time in a completely connected network. Every PE has its local memory space to store data required to execute programs. The communication model among PEs is based on a message passing model such as MPI [13]. In such a communication model, it is assumed that multiple data are communicated at the same time with a constant communication bandwidth. This means that communication bandwidth is time invariant[1].

In chapter 3, we assume the number of PEs is unbounded. On the other hand, in chapter 4, there are limited number of PEs and each PE has a non-identical processing speed and a non-identical communication bandwidth. The set of PEs is expressed as $P = \{P_1, P_2, \ldots, P_m\}$, and let the set of processing speeds as α, i.e.,

$$\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}. \tag{2.2}$$

Let the set of communication bandwidths as β, i.e.,

$$\beta = \begin{pmatrix} \infty & \beta_{1,2} & \beta_{1,3} & \ldots & \beta_{1,m} \\ \beta_{2,1} & \infty & \beta_{2,3} & \ldots & \beta_{2,m} \\ \beta_{3,1} & \beta_{3,2} & \infty & \ldots & \beta_{3,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \beta_{m,1} & \beta_{m,2} & \beta_{m,3} & \ldots & \infty \end{pmatrix}. \tag{2.3}$$

$\beta_{i,j}$ means the communication bandwidth from P_i to P_j. The processing time in the case that n_k^s is processed on P_i is expressed as $t_p(n_k^s, \alpha_i) = w(n_k^s)/\alpha_i$. The data transfer time of $e_{k,l}^s$ over $\beta_{i,j}$ is $t_c(e_{i,j}^s, \beta_{k,l}) = c(e_{i,j}^s)/\beta_{k,l}$. This means that both processing time and data transfer time are not changed with time, and suppose that data transfer time within one PE is negligible. Since every processing speed and communication bandwidth are the same among PEs in homogeneous distributed systems, without loss of generality both α_i and $\beta_{i,j}$ are set to 1, i.e., $w(n_k^s)$ and $c(e_{i,j}^s)$ are considered as the processing time and the data transfer time. Such assumptions are applied in chapter 3.

2.4 Schedule length

Each task can be executed after every data from its immediate predecessors have been arrived. At first, let the start time (scheduled time) of n_j^s be $t_s(n_j^s)$, and let the completion time of n_j^s be $t_f(n_j^s)$. Then $t_f(n_j^s)$ is defined as follow.

$$t_f(n_j^s) \quad = \quad t_s(n_j^s) + t_p(w(n_j^s), \alpha_p). \tag{2.4}$$

[1]Since our study is based on the classical communication model, we do not focus on multiple data transmission at one time [40], one-port model [40]

When every data from $pred(n_j^s)$ has been arrived, n_j^s can be executed immediately. However, even if every data from $pred(n_j^s)$ has been arrived at n_j^s, n_j^s can not be started until the execution of another task in the same cluster is finished.

In this dissertation, the time when every data from every immediate predecessor tasks has been arrived is named as Data Ready Time (DRT [16, 41]). DRT of $n_j^s (n_j^s \in cls_s(k))$ is defined as follow.

$$\begin{aligned} t_{dr}(n_j^s) &= \max_{n_i^s \in pred(n_j^s)} \left\{ t_f(n_i^s) + t_c(c(e_{i,j}^s), \beta_{p,q}) \right\} \\ &= \max \left\{ \max_{\substack{n_i^s \in pred(n_j^s), \\ n_i^s \in cls_s(k)}} \left\{ t_f(n_i^s) \right\}, \max_{\substack{n_i^s \in pred(n_j^s), \\ n_i^s \notin cls_s(k)}} \left\{ t_f(n_i^s) + t_c(c(e_{i,j}^s, \beta_{p,q})) \right\} \right\}. \end{aligned} \quad (2.5)$$

From eq.(2.5), it can be seen that $t_{dr}(n_j^s)$ is derived from the maximum completion time of a task in $pred(n_j^s)$ included in the same cluster and the maximum data arrival time from a task in $pred(n_j^s)$ included in other clusters. In the former case, data transfer time between $pred(n_j^s)$ and n_j^s is zero, because they are included in the same cluster. On the other hand, the latter case requires data transfer time of $c(e_{i,j}^s)$. If execution of every immediate predecessor task has been completed but every data from one or more tasks from other clusters has not arrived, the task must be wait for delay its execution. In such a case, the data waiting time exists. The data waiting time of n_j^s in $cls_s(k)$ is defined as $I(n_j^s, k)$, which is derived as follow.

$$I(n_j^s, k) = \begin{cases} 0, \ if \max\limits_{\substack{n_i^s \in pred(n_j^s), \\ n_i^s \in cls_s(k)}} \{t_f(n_i^s)\} \geq \max\limits_{\substack{n_i^s \in pred(n_j^s), \\ n_i^s \notin cls_s(k)}} \left\{ t_f(n_i^s) + t_c(c(e_{i,j}^s), \beta_{p,q}) \right\}, \\ \max\limits_{\substack{n_i^s \in pred(n_j^s), \\ n_i^s \notin cls_s(k)}} \left\{ t_f(n_i^s) + t_c(c(e_{i,j}^s), \beta_{p,q}) \right\} - \max\limits_{\substack{n_i^s \in pred(n_j^s), \\ n_i^s \in cls_s(k)}} \left\{ t_f(n_i^s) \right\}, \ otherwise. \end{cases} \quad (2.6)$$

Even if $t_{dr}(n_j^s)$ is known, the scheduled time of n_j^s may be varied by the execution order (i.e., the scheduling policy) when some tasks are independent from n_j^s in the same cluster have not been scheduled. Thus, we have $t_{dr}(n_j^s) \leq t_s(n_j^s)$ [16, 41]. From the relationship, it follows the fact that the completion time of each task depends on the scheduling policy. In this dissertation, the schedule length of G_{cls}^s is defined as $sl(G_{cls}^s)$.

$$sl(G_{cls}^s) = \max_{n_j^s \in cls_s(k), cls_s(k) \in V_{cls}^s} \left\{ t_f(n_j^s) \right\}. \quad (2.7)$$

In eq.(2.7), it is assumed that the start time of a START task is set to 0.

Chapter 3

Task clustering in homogeneous distributed systems

3.1 Introduction

In distributed systems where each PE sends or receives data over the network, scheduling tasks to minimize schedule length is very important [12, 18, 27–29]. It is known that a task graph which consists of a DAG, has been considered as an NP-complete problem [27]. If the number of PEs is given and every PE must be used, a priority for scheduling order is adopted by conventional approaches. Such approaches have been named as "list scheduling." On the other hand, if the number of PEs is not given or not all PEs must be used, we must derive not only execution order for each task, but also the number of PEs in order to obtain a good schedule length. As one approach in such a case, task clustering [17] has been known. One fundamental feature of a task clustering is to merge several tasks into one cluster by localizing communication overhead among them, so that each cluster corresponds to each assignment unit per one PE. However, if the smaller communication overhead among tasks becomes, the longer schedule length becomes, thereby the schedule length can be prolonged. In distributed systems such as grid where several application programs can be processed simultaneously, effectively using computational resources is a key factor to achieve short schedule lengths for all executing applications. To achieve efficient utilization of computational resources, it is important to derive the execution order to minimize schedule length, while to reduce the number of clusters as much as possible. There are several heuristic approaches whose purposes are to reduce the number of clusters by merging several clusters into a larger one after a task clustering. Pyrros compiling infrastructure [25] adopts a criterion for equalizing each cluster size. Liou et al. proposed two task merging approaches, i.e., LB(Load Balancing) and CTM (Communication Traffic Minimizing) [26]. The merging criterion of LB is the same as the cluster merging adopted in Pyrros [25] except that LB does not consider data dependencies among tasks. On the other hand, the criterion of cluster merging performed by CTM is that sum of data transfer time among clusters to be merged is minimized as much as possible. According to the results of performance comparisons between LB and CTM, both of cluster merging approaches have bad effects on the schedule length when they are performed after a task clustering,

e.g., CASS-II [26]. According to the literature [26], by using LB after CASS-II, the schedule length is prolonged up to 19 % compared with the schedule length obtained by only CASS-II. In the case of performing CTM after CASS-II, the schedule length is prolonged up to 55 %. This means that a cluster merging approach which sacrifices task parallelism can prolong the schedule length. Thus, a cluster merging strategy for maintaining task parallelism with the small number of clusters is required.

In this chapter, we present a cluster size determination method in order to obtain a good schedule length with the small number of clusters in homogeneous distributed systems. As one heuristic for reducing the number of clusters generated by a task clustering, we impose the lower bound "δ" of every cluster size.

The fundamental objective is to minimize the schedule length. Hence, we derived the lower bound of every cluster size while effectively minimizing the schedule length. Then we present requirements and the algorithm for a task clustering heuristic, and then experimental comparisons by simulations are presented.

3.2 Problem definition and assumptions

3.2.1 Cluster merging

If the number of generated clusters by a task clustering is smaller than that of actual existing PEs, every cluster can be assigned to a PE. Otherwise, it is necessary to reduce the number of clusters by merging them such that each PE can be assigned to one cluster [25,26]. In this dissertation, cluster merging means to a procedure for merging several clusters generated by a task clustering.

Figure 3.1 shows an example of a task clustering and cluster mergings. In this figure, (a) represents the initial state of the DAG, and (b) represents the state after a task clustering has been finished. The schedule length in (a) is equal to the critical path length (the maximum path length including both every task size and every data size on a path), i.e., 23 by tracing $n_1^0 \rightarrow n_3^0 \rightarrow n_5^0 \rightarrow n_7^0$. On the other hand, in (b), no task scheduling is required because every cluster is linear. Hence, the schedule length in (b) is uniquely determined to be 20.

If the number of clusters must be reduced to two due to the fact that there are only two PEs, a cluster merging such as figure 3.1 (c), (d), and (e) is needed. As for each cluster generated in (b), if $cls_4(1)$ and $cls_4(2)$ are merged by LB [26], some independent tasks exist, e.g., "n_2^5 and n_3^5" and "n_2^5 and n_5^5" in (c), (d) and (e). As a result, the schedule length in the order of $n_2^5 \rightarrow n_3^5 \rightarrow n_5^5$ in (c) is 23. In (d), the schedule length in the order of $n_3^5 \rightarrow n_2^5 \rightarrow n_5^5$ is 24, because the data arrival time of $e_{2,7}^5$ at n_7^5 is delayed by the increase of the start time of n_2^5. In (e), the schedule length is larger than that of (c) and (d) by scheduling n_2^5 in $cls_5(2)$ at the latest execution order. From those examples, it can be concluded that the schedule length after a cluster merging can become larger than that after a task clustering, because the number of tasks which can be executed at the same time may be reduced by a cluster merging. It can also be concluded that the schedule length is varied depending on the execution order for each task, even if the set of tasks belonging to the cluster is same among (c), (d), and (d).

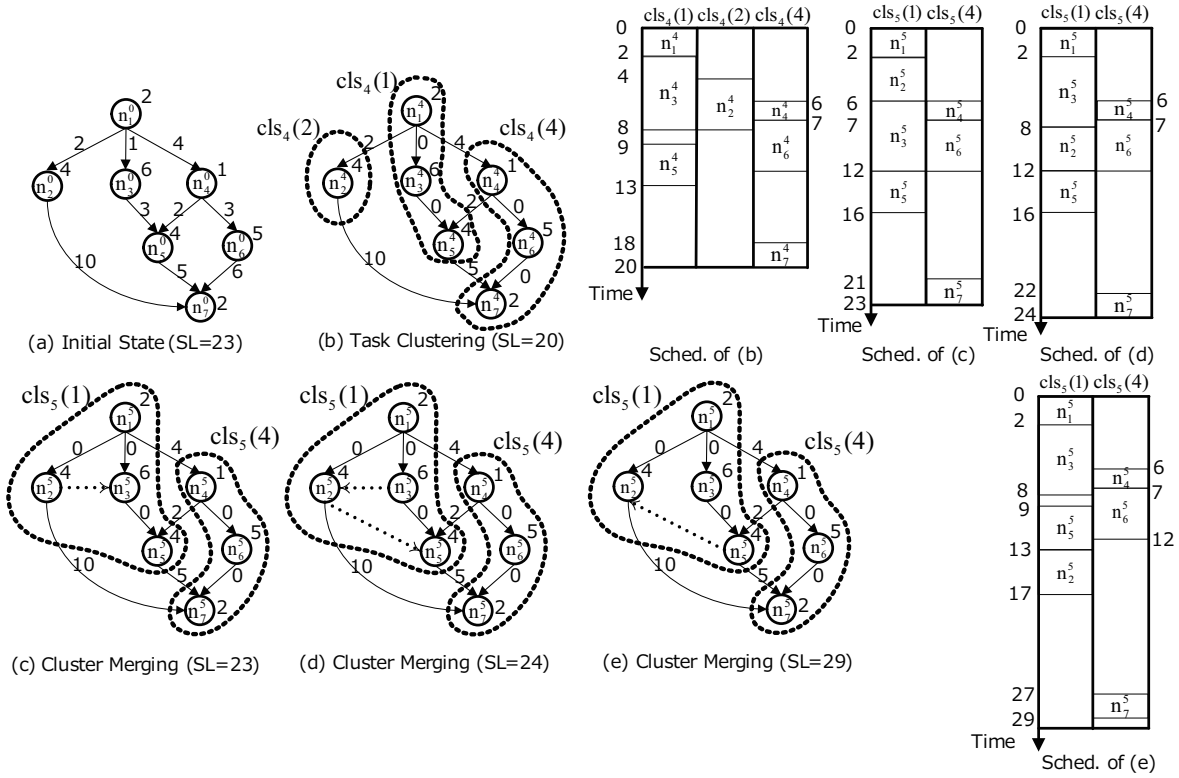


Figure 3.1: Derivation of the schedule length by task clustering and cluster merging (SL: Schedule Length)
(appears in [22]).

3.2.2 Problems in conventional approaches

The objective of conventional task clustering heuristics [17, 18, 24, 27, 29–33] mainly derive the number of clusters (PEs) when the schedule length is minimized. In such approaches, some heuristics adopt criteria that a task merging step is not accepted if the schedule length is increased, otherwise the task merging step is accepted [18, 27, 29, 30]. In Convex Clustering [32], tasks having the "convex" relationship each other are selected for a task merging step. Thus, only precedence relationships are considered in Convex Clustering. Other approaches are based on the optimization methods [31,33]. Those approaches described above do not impose any constraint for limiting the number of clusters to be generated, so that the number of obtained clusters may become huge depending on not only each cluster structure, but also criteria for each task merging step. If the objective of the task clustering is to minimize the schedule length, the larger the number of generated clusters is, the lower the speed up ratio (the degree of contribution for reducing the schedule length) per one PE may become.

As for the cluster merging, in the literature [25] mentions "Cluster Merging(CM)," which performs cluster mergings based on the criterion that every cluster size is equalized until the number of clusters is equal to that of PEs, while it does not take the precedence relationship

between tasks. As a result, several tasks without precedence relationships belong to the same cluster, so that the schedule length can be larger like figure 3.1(c), (d), and (e).

Load Balancing(LB) described in the literature [26] and Communication Traffic Minmizing (CTM) [26] are cluster merging heuristics which are performed after one of task clustering heuristic, i.e., CASS-II [29]. Those two cluster mergings try to merge two clusters, in which at least one precedence relationship exists. However, LB and CTM are performed after a task clustering, so that a cluster required for cluster merging can have two or more tasks. Thus, some tasks may have no precedence relationship in a merged cluster like figure 3.1(c), (d), and (e).

With those points described above, it can be said that conventional approaches have two problem as follows.

- The number of required PEs derived by a conventional task clustering exceeds the number of actual PEs.
- There is no criterion for minimizing the increase of the schedule length by a cluster merging.

3.2.3 Proposal

The objective of our proposal is to minimize the schedule length with the small number of PEs. Our proposals are as follows.

1. **Derivation of the lower bound of every cluster size (defined as δ_{opt}), by which the schedule length can be minimized.**
   From figure 3.1 (c) and (d), it can be said that the schedule length after a cluster merging is longer due to the fact that independent tasks are included in the same cluster. Thus, a criterion for minimizing the schedule length with the small number of clusters is needed. In this chapter, the lower bound of each cluster size is imposed as δ for limiting the number of PEs, and we study how to decide δ for minimizing the schedule length. Then we derive δ_{opt}, which is the value of δ when the schedule length can be minimized.
2. **The policies for task merging steps under the constraint that every cluster size is δ_{opt} or more.**
   Even if δ_{opt} is decided before a task merging step, the schedule length can be varied by a task merging policy and an execution order for each task. We present a task clustering algorithm, which performs task merging steps until each cluster size is δ_{opt} or more and which tries to minimized the schedule length.

3.3 Derivation of the lower bound of every cluster size

In this section, we present details about the first proposal in sec.3.2.3. For G_{cls}^{R}, which is the state after R task merging steps have been performed, the following condition is assumed to be satisfied.

$$w(cls_R(i)) \geq \delta \ s.t., \ \forall cls_R(i) \in V_{cls}^R. \tag{3.1}$$

Eq.(3.1) means that every cluster size is δ or more. When $s < R$, at least one cluser size is smaller than δ. In this chapter, the range of s is defined as $0 \leq s \leq R$. In this section, we study how to decide δ for minimizing the schedule length.

3.3.1 Policies for deriving the lower bound of every cluster size

If each task in V_{cls}^s is scheduled, the schedule length varies by the execution order for each task. Even if the scheduling policy is decided, the schedule length is also changed by the combination of tasks in each cluster. This fact leads to that to find the optimal schedule is said to be NP-hard problem [27]. Furthermore, each cluster can not be generated before the lower bound has been derived. This means that each combination in a cluster, the execution order for each task, and DRT of each task (defined at eq.(2.6)) are unknown before a task clustering, and the schedule length is also unknown before the lower bound of every cluster size has been decided. Hence, it is necessary that an indicative value, which can have effect on the schedule length by varying the lower bound of every cluster size, must be derived. Thus, we define the indicative value as $sl_w(G_{cls}^s)$, which is the maximum schedule length, provided that each task is executed as late as possible without data waiting time (defined in sec.3.3.2). We define the lower bound of every cluster size when the upper bound of $sl_w(G_{cls}^R)$ is minimized as δ_{opt}(defined in sec.3.3.7). In sec.3.3.8, how the variation of $sl_w(G_{cls}^R)$ has effect on the actual schedule length is described.

3.3.2 Definition of WSL

Table 3.1 shows notations and definitions for deriving $sl_w(G_{cls}^s)$. In the i-th cluster, we define the set of tasks which can firstly be executed as $top_s(i)$. The set of tasks in the i-th cluster which requires incoming data communication from other clusters is defined as $in_s(i)$. On the other hand, The set of tasks which requires outgoing data communication with other clusters is defined as $out_s(i)$, and the set of tasks which can be executed at the last in the i-th cluster is defined as $btm_s(i)$.

More specifically, $top_s(i)$ is the set of tasks whose all immediate predecessor tasks belong to other clusters. Every task in $in_s(i)$ has at least one immediate predecessor task which belongs to another cluster. Every task in $out_s(i)$ has at least one immediate successor task which belongs to another cluster. Every task in $btm_s(i)$ has one or more successor tasks all of which belong to other clusters. Thus, every task in $btm_s(i)$ is included in $out_s(i)$.

$desc(n_k^s, i)$ is the union of the set of tasks executed after n_k^s in $cls_s(i)$ and n_k^s itself. $S(n_k^s, i)$ is the sum of task size which can be executed before n_k^s, provided that n_k^s is scheduled as late as possible in $cls_s(i)$. Next, we define $tlevel(n_k^s)$, which is the scheduled time of n_k^s when $cls_s(i)$ is executed as late as possible. If $n_k^s \in top_s(i)$(where $n_k^s \in cls(i)$), $tlevel(n_k^s)$ is the time every data from immediate predecessor tasks of n_k^s arrives, since those immediate predecessor tasks belong to other clusters. $TL_s(i)$ is the maximum value of $tlevel(n_k^s)$, where $n_k^s \in top_s(i)$. If $TL_s(i) = tlevel(n_k^s)$ for $n_k^s \in top_s(i)$, every task (except n_k^s) which belongs to $top_s(i)$ is not executed until the completion time of n_k^s.

Table 3.1: Parameter definition which is related to $sl_w(G^s_{cls})$ ($n^s_k \in cls_s(i)$) (appears in [22]).

Parameter	Definition
$top_s(i)$	$\{n^s_k \mid \forall n^s_l \in pred(n^s_k) s.t., n^s_l \notin cls_s(i)\} \cup$ START Tasks $\in cls_s(i)$.
$in_s(i)$	$\{n^s_k \mid \exists n^s_l \in pred(n^s_k) s.t., n^s_l \notin cls_s(i)\} \cup$ START Tasks $\in cls_s(i)$.
$out_s(i)$	$\{n^s_k \mid \exists n^s_l \in suc(n^s_k) s.t., n^s_l \notin cls_s(i)\} \cup$ END Tasks $\in cls_s(i)$.
$btm_s(i)$	$\{n^s_k \mid \forall n^s_l \in suc(n^s_k), s.t., n^s_l \notin cls_s(i)\} \cup$ END Tasks $\in cls_s(i)$.
$desc(n^s_k, i)$	$\{n^s_l \mid n^s_k \prec n^s_l, n^s_l \in cls_s(i)\} \cup \{n^s_k\}$
$S(n^s_k, i)$	$\sum_{n^s_l \in cls_s(i)} w(n^s_l) - \sum_{n^s_l \in desc(n^s_k, i)} w(n^s_l)$
$tlevel(n^s_k)$	$\begin{cases} \max\limits_{n^s_l \in pred(n^s_k)} \{tlevel(n^s_l) + w(n^s_l) + c(e_{l,k})\}, \ where\ n^s_k \in top_s(i), \\ TL_s(i) + S(n^s_k, i), \ otherwise. \end{cases}$
$TL_s(i)$	$\max\limits_{n^s_k \in top_s(i)} \{tlevel(n^s_k)\}$
$blevel(n^s_k)$	$\max\limits_{n^s_l \in suc(n^s_k)} \left\{ w(n^s_k) + c(e^s_{k,l}) + blevel(n^s_l) \right\}$
$level(n^s_k)$	$tlevel(n^s_k) + blevel(n^s_k)$
$BL_s(i)$	$\max\limits_{n^s_k \in out_s(i)} \{S(n^s_k, i) + blevel(n^s_k)\}$
$LV_s(i)$	$TL_s(i) + BL_s(i) = \max\limits_{n^s_k \in cls_s(i)} \{level(n^s_k)\}$
$sl_w(G^s_{cls})$	$\max\limits_{cls_s(i) \in V^s_{cls}} \{LV_s(i)\}$

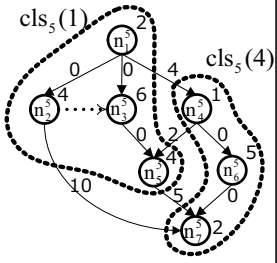
Next, we define $tlevel(n^s_k)$ in the case of $n^s_k \notin top_s(i)$. From eq.(2.6), the data waiting time $I(n^s_k, i)$ for n^s_k depends on the scheduling policy(i.e., execution order of every task). That is, $I(n^s_k, i)$ is unknown before a task clustering. Thus, we define $tlevel(n^s_k) = TL_s(i) + S(n^s_k, i)$ for $n^s_k \notin top_s(i)$, which means that the start time of n^s_k when $I(n^s_k, i)$ is neglected. Let $blevel(n^s_k)$ the maximum path length from n^s_k to the END task. That is, $blevel(n^s_k)$ is the maximum value of the time taken if every task which has precedence relationships with n^s_k from n^s_k to the END task is executed. $BL_s(i)$ is the time taken from the start time of a task in $cls(i)$ to the completion time of the END task, i.e., the maximum of the sum of $S(n^s_k, i)$ and $blevel(n^s_k)$. $LV_s(i)$ is the sum of $TL_s(i)$ and $BL_s(i)$. If we define

$$level(n^s_k) = tlevel(n^s_k) + blevel(n^s_k),$$

we have

$$\begin{aligned} LV_s(i) &= TL_s(i) + BL_s(i) = TL_s(i) + \max_{n^s_k \in out_s(i)} \{S(n^s_k, i) + blevel(n^s_k)\} \\ &= \max_{n^s_k \in out_s(i)} \{TL_s(i) + S(n^s_k, i) + blevel(n^s_k)\} \\ &= \max_{n^s_k \in cls_s(i)} \{level(n^s_k)\}. \end{aligned} \tag{3.2}$$

For each cluster $cls_s(i) \in V^s_{cls}$, the maximum of $LV_s(i)$ is $sl_w(G^s_{cls})$. From this value, it can be seen that $sl_w(G^s_{cls})$ is derived by deciding which cluster takes the maximum of LV, i.e., which task in a cluster is scheduled as late as possible.



$top_5(1) = \{n_1^5\}$, $top_5(4) = \{n_4^5\}$,
$in_5(1) = \{n_1^5, n_5^5\}$, $in_5(4) = \{n_4^5, n_7^5\}$
$out_5(1) = \{n_1^5, n_2^5, n_5^5\}$, $out_5(4) = \{n_4^5, n_7^5\}$
$btm_5(1) = \{n_2^5, n_5^5\}$, $btm_5(4) = \{n_7^5\}$
------ $cls_5(1)$ section------
$TL_5(1) = tlevel(n_1^5) = 0$,
$tlevel(n_2^5) = TL_5(1) + S(n_2^5, 1) = 0 + w(n_1^5) + w(n_3^5) + w(n_5^5) = 12$,
$blevel(n_2^5) = w(n_2^5) + c(e_{2,7}^5) + blevel(n_7^5) = 4 + 10 + 2 = 16$,
$level(n_2^5) = 28$,
$tlevel(n_3^5) = TL_5(1) + S(n_3^5, 1) = 0 + w(n_1^5) + w(n_2^5) + w(n_5^5) = 10$
$S(n_1^5, 1) = 0$,
$blevel(n_1^5) = w(n_1^5) + \max\{blevel(n_2^5), blevel(n_3^5), 4 + blevel(n_4^5)\}$
$= 2 + 4 + blevel(n_4^5) = 20$, $level(n_1^5) = 20$
$tlevel(n_5^5) = TL_5(1) + S(n_5^5, 1) = 0 + w(n_1^5) + w(n_2^5) + w(n_3^5) = 12$
$blevel(n_5^5) = 4 + 5 + 2 = 11$, $level(n_5^5) = 23$
$BL_5(1) = \max_{n_k^5 \in out_5(1)} \{S(n_k^5, 1) + blevel(n_k^5)\}$
$= S(n_2^5, 1) + blevel(n_2^5) = 28$
$LV_5(1) = TL_5(1) + BL_5(1) = 28 = level(n_2^5)$

------ $cls_5(4)$ section------
$TL_5(4) = tlevel(n_4) = 2 + 4 = 6$,
$tlevel(n_6) = TL_5(4) + w(n_4^5) = 6 + 1 = 7$
$blevel(n_4^5) = w(n_4^5) + \max\{2 + blevel(n_5^5), blevel(n_6^5)\}$
$= 1 + 13 = 14$
$level(n_4^5) = 6 + 14 = 20$
$tlevel(n_7^5) = TL_5(4) + w(n_4^5) + w(n_6^5) = 6 + 1 + 5 = 12$
$blevel(n_7^5) = 2$, $level(n_7^5) = 2 + 12 = 14$
$BL_5(4) = \max_{n_k^5 \in out_5(4)} \{S(n_k^5, 4) + blevel(n_k^5)\}$
$= S(n_4^5, 4) + blevel(n_4^5) = 0 + 14 = 14$
$LV_5(4) = TL_5(4) + BL_5(4) = 6 + 14 = 20 = level(n_4^5)$
------ $cls_5(4)$ section END------
$sl_w(G_{cls}^5) = \max\{LV_5(1), LV_5(4)\} = LV_5(1) = 28$
------ $cls_5(1)$ section END------

Figure 3.2: Example of each defined symbols
(appears in [22]).

Example 1. *Figure 3.2 shows an example for deriving $sl_w(G_{cls}^5)$. The DAG in this figure is identical to that of 3.1 (c),(d) and (e). In figure 3.2, $LV_5(1)$ is equal to $level(n_2^5)$, and $LV_5(4)$ is equal to $level(n_4^5)$. From this fact, it can be said that the schedule length becomes large when n_2^5 is scheduled as the last task in $cls_5(1)$. On the other hand, the execution order in $cls_5(4)$ is unique because $cls_5(4)$ is linear. Since $LV_5(1) = 28$ and $LV_5(4) = 20$, we have $sl_w(G_{cls}^5) = LV_5(1)$. If the data waiting time (defined at eq.(2.6)) is neglected, the maximum schedule length is obtained when n_2^5 is scheduled as late as possible in $cls_5(1)$. This fact holds in figure 3.1 (e). $sl(G_{cls}^5) = 29$ in figure 3.1 (e), while $sl_w(G_{cls}^5) = 28$ because the data waiting time($9 - 8 = 1$ unit time at 3.1 (e)) from n_4^5 is neglected at n_5.* ■

3.3.3 Preliminary for the analysis of WSL

Since $sl_w(G_{cls}^R)$ is the value decided after R task merging steps (R is defined by eq.(3.1)), $sl_w(G_{cls}^R)$ is unknown before a task clustering. Thus, we study the upper bound of $sl_w(G_{cls}^R)$, for different cluster structure (precedence relationships among tasks in a cluster and the set of tasks in a cluster). In particular, we try to find the lower bound of the cluster size when the upper bound of $sl_w(G_{cls}^s)$ is minimized. Then we determin the objective of our task clustering by clearify the relationship between $sl_w(G_{cls}^R)$ and the schedule length.

Table 3.2 shows definitions for the analysis of the upper bound of $sl_w(G_{cls}^R)$. Let the set of tasks included in the execution path be seq_s, by which $sl_w(G_{cls}^s)$ is decided after $s(s \leq R)$ task merging steps. That is, seq_s is the union of the set of tasks by which $tlevel(n_k^R)$ is decided for n_k^R such that $sl_w(G_{cls}^s) = LV_R(i) = level(n_k^R)$ and the set of tasks which belong to the path by which $blevel(n_k^R)$ is decided (detailed in example 2).

Next, in seq_s let the path p in which every task has precedence relationships be $seq_s^{\prec}$. $seq_s^{\prec}$ is the union of tasks and edges in a path p, in which n_k^s, n_l^s and $e_{k,l}^s$ satisfy the following

Table 3.2: Parameter definitions which are used in analysis on $sl_w(G^s_{cls})$ $(0 \leq s \leq R)$ (appears in [22]).

Parameter	Definition
seq_s	Set of tasks by which $sl_w(G^s_{cls})$ is derived.
p	One path of G^0_{cls}, i.e., $\{n^0_0, n^0_1, n^0_2, \ldots, n^0_k\} \cup \{e^0_{0,1}, e^0_{1,2}, \ldots e^0_{k-1,k}\}$, by which a sequence $< n^0_0, n^0_1, n^0_2, \ldots, n^0_k >$ is constructed, where $e^0_{l-1,l} \in E_0$, n^0_0 is a START task and n^0_k is an END task.
p'	One subpath of G^0_{cls}, i.e., $\{n^0_0, n^0_1, n^0_2, \ldots, n^0_k\} \cup \{e^0_{0,1}, e^0_{1,2}, \ldots e^0_{k-1,k}\}$, by which a sequence $< n^0_0, n^0_1, n^0_2, \ldots, n^0_k >$ is constructed, where $e^0_{l-1,l} \in E_0$, n^0_0 is not a START task or n^0_k is not an END task.
$seq^{\prec}_s$	One path in which every task belongs to seq_s.
$seq^{\prec}_s(i)$	Set of subpaths in each of which every task in $cls(i)$ belongs to $seq^{\prec}_s$.
w_{max}	$\max_{cls_R(i) \in V^R_{cls}} \{w(cls_R(i))\} - \delta$
ϕ	♯ of clusters in which at least one task belongs to $seq^{\prec}_s$.
cp	Critical path length of G^0_{cls}.
cp_w	$\max_{p \in G^0_{cls}} \left\{ \sum_{n^0_k \in p} w(n^0_k) \right\}$.
$g_{\min}$ [16,24]	$\min_{n^0_k \in V^0_{cls}} \left\{ \frac{\min_{n^0_j \in pred(n^0_k)} \{w(n^0_j)\}}{\max_{n^0_j \in pred(n^0_k)} \{c(e^0_{j,k})\}}, \frac{\min_{n^0_l \in suc(n^0_k)} \{w(n^0_l)\}}{\max_{n^0_l \in suc(n^0_k)} \{c(e^0_{k,l})\}} \right\}$.
$g_{\max}(n^s_k)$	$\max \left\{ \frac{\max_{n^0_j \in pred(n^0_k)} \{w(n^0_j)\}}{\min_{n^0_j \in pred(n^0_k)} \{c(e^0_{j,k})\}}, \frac{\max_{n^0_l \in suc(n^0_k)} \{w(n^0_l)\}}{\min_{n^0_l \in suc(n^0_k)} \{c(e^0_{k,l})\}} \right\}$.
Δsl_w	$sl_w(G^R_{cls}) - sl_w(G_{org}) = sl_w(G^R_{cls}) - cp$.
$\Delta sl_{w,up}$	An upper bound of Δsl_w.
$len(seq^{\prec}_s(i))$	$\sum_{n^s_k \in seq^{\prec}_s(i)} w(n^s_k) + \sum_{\substack{n^s_k, n^s_l \in seq^{\prec}_s(i), \\ e^0_{k,l} \in E_0}} c(e^0_{k,l})$.
$len(seq^{\prec}_s)$	$\sum_{n^s_k \in seq^{\prec}_s} w(n^s_k) + \sum_{\substack{n^s_k, n^s_l \in seq^{\prec}_s, \\ e^0_{k,l} \in E_0}} c(e^0_{k,l})$.
ΔL_i	$\sum_{n^R_k \in cls_R(i) \cap seq_R} w(n^R_k) - len(seq^{\prec}_R(i))$.
ΔL_w	$sl_w(G^R_{cls}) - len(seq^{\prec}_R)$.
$\Delta L_{w,up}$	An upper bound of ΔL_w.

condition.

$$n^s_k \in seq_s, n^s_l \in seq_s, n^s_k \in pred(n^s_l).$$

Note that more than one of $seq^{\prec}_s$ can exist for each task in seq_s, because $seq^{\prec}_s$ is defined by tracing tasks with precedence relationships. For example, if seq_s is $\{n^s_1, n^s_2, n^s_3, n^s_4\}$ and there are two paths, e.g., $\{n^s_1, n^s_2, n^s_4\} \cup \{e^s_{1,2}, e^s_{2,4}\}$ and $\{n^s_1, n^s_3, n^s_4\} \cup \{e^s_{1,3}, e^s_{3,4}\}$, those two paths are one of $seq^{\prec}_s$, respectively.

In $seq^{\prec}_s$, let $seq^{\prec}_s(i)$ be the union of the set of tasks belonging to $cls_s(i)$ and the set of edges among them. That is, $seq^{\prec}_s(i)$ is the subset of the path p'(defined in table 3.2) and satisfying the following condition. n^s_k, n^s_l in $seq^{\prec}_s(i)$ are expressed as follow.

$$n^s_k \in seq^{\prec}_s, n^s_l \in seq^{\prec}_s, n^s_k \in pred(n^s_l), n_k, n_l \in cls(i).$$

$seq^{\prec}_s(i)$ is a subset of $seq^{\prec}_s$. Thus, there is only one $seq^{\prec}_s(i)$ in $seq^{\prec}_s$. If there are two or more $seq^{\prec}_s$, two or more $seq^{\prec}_s(i)$ exist.

The schedule length of G^0_{cls} is the critical path length, which is expressed as cp. The maximum of the summation for every task size on a path is expressed as cp_w.

Let $g_{max}(n^0_k)$ be the maximum granularity of n^0_k, and let g_{min} be the minimum granularity for every task in G^0_{cls} [24].

Example 2. *In this example, the meanings of seq_s and $seq^{\prec}_s$ are described. From figure 3.2, we have $sl_w(G^5_{cls}) = level(n^5_2)$. Since $\{n^5_1\}$ is executed before n^5_2 in $cls_5(1)$ and $\{n^5_1\}$ has the precedence relationship with n^5_2, we have $seq^{\prec}_5(1) = \{n^5_1, n^5_2\} \cup \{e^5_{1,2}\}$. Furthermore, we have*

$$blevel(n^5_2) = w(n^5_2) + c(e^5_{2,7}) + blevel(n^5_7), seq^{\prec}_5(4) = \{n^5_7\}.$$

Hence, the following result is obtained.

$$seq^{\prec}_5 = \{n^5_1, n^5_2, n^5_7\} \cup \{e^5_{1,2}, e^5_{2,7}\}.$$

On the other hand, in $cls_5(1)$, any one of $\{n^5_1, n^5_3, n^5_5\}$ can be executed before n^5_2. Since we have

$$blevel(n^5_2) = w(n^5_2) + c(e^5_{2,7}) + blevel(n^5_7)$$

as described above, we have $seq_5 = \{n^5_1, n^5_2, n^5_3, n^5_5, n^5_7\}$. ■

Example 3. *Figure 3.3 is another example for deriving seq_s and $seq^{\prec}_s$. In this figure, at (1) $cls_s(i)$ and $cls_s(i+1)$ are linear, and (2) shows the cluster structure in the case of $sl_w(G^s_{cls}) = level(n^s_9)$ with both $cls_s(i)$ and $cls_s(i+1)$ being non-linear. Furthermore, dashed lines mean the execution order of tasks dominating $sl_w(G^s_{cls})$.*

At (1), execution orders in $cls_s(i+1)$ and $cls_s(i)$ are unique because those clusters are linear. As a result, $\{n^s_1, n^s_2, n^s_3, n^s_4, n^s_5\}$ belong to both seq_s and $seq^{\prec}_s$. (2) shows that tasks in both dashed arrows and dashed lines belong to seq_s. Here, if $sl_w(G^s_{cls}) = level(n^s_9)$, we have the following result.

$$\begin{aligned} tlevel(n^s_9) &= tlevel(n^s_5) + w(n^s_6) + w(n^s_8) + w(n^s_7), \\ blevel(n^s_9) &= w(n^s_9) + c(e^s_{9,11}) + blevel(n^s_{11}). \end{aligned}$$

The start time of n^s_4 is delayed as late as possible by scheduling n^s_4 after n^s_2, n^s_3 in the dashed circle, i.e., $tlevel(n^s_4) = tlevel(n^s_1) + w(n^s_2) + w(n^s_3)$. Since every task in n^s_1, n^s_2, n^s_4 has precedence relationships each other, $seq^{\prec}_s(i)$ is $\{n^s_1, n^s_2, n^s_4\} \cup \{e^s_{1,2}, e^s_{2,4}\}$.

In $cls_s(i+1)$, since every task in n^s_5, n^s_7, n^s_9 has precedence relationships each other and belongs to seq_s, we have $seq^{\prec}_s(i+1) = \{n^s_5, n^s_7, n^s_9\} \cup \{e^s_{5,7}, e^s_{7,9}\}$. ■

3.3.4 Relationship between δ and WSL after R task merging steps

As described in sec.3.2, only one task belongs to a cluster in G^0_{cls} and then we obtain the schedule length of $G^0_{cls} = G_{org}$ as cp. Moreover, since $\{n^0_k\} = cls_0(i) \in V^0_{cls}$ in $G^0_{cls} = G_{org}$, from eq.(3.2) we have

$$\begin{aligned} sl_w(G_{org}) &= sl_w(G^0_{cls}) = \max_{cls_0(i) \in V^0_{cls}} \{LV_0(i)\} \\ &= \max_{n^0_k \in V_0} \left\{level(n^0_k)\right\} = \max_{n^0_k \in V_0} \left\{tlevel(n^0_k) + blevel(n^0_k)\right\}. \end{aligned} \tag{3.3}$$

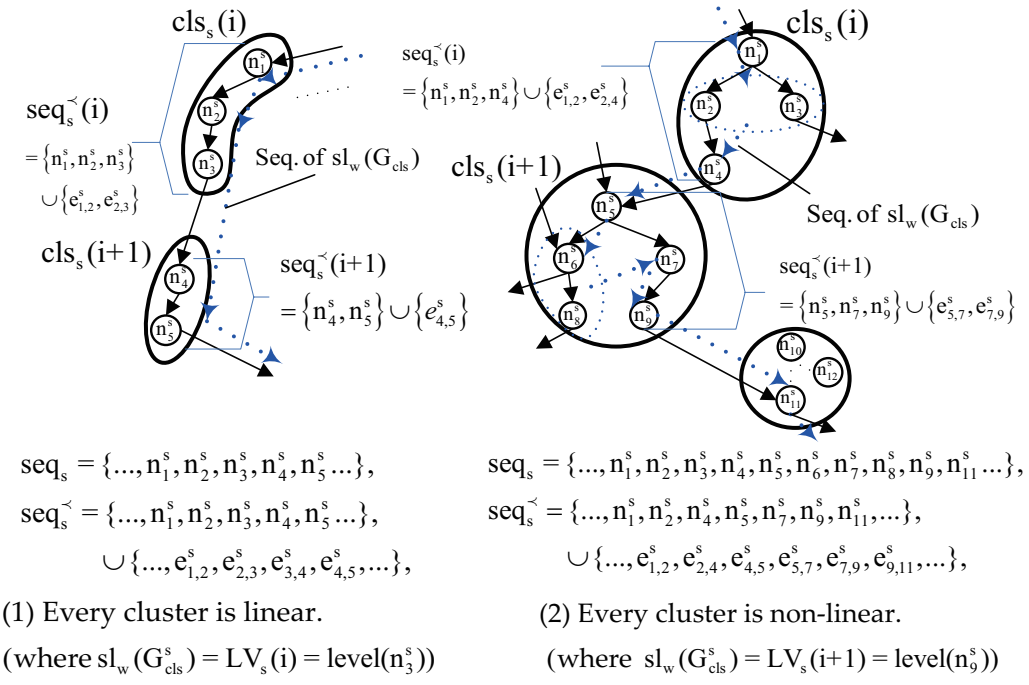


Figure 3.3: Concept of the upper bound of $sl_w(G^s_{cls})$
(appears in [22]).

From definitions of $tlevel(n_k^R)(n_k^R \in top_R(i))$ and $blevel(n_k^R)$ in table 3.1, with eq.(3.3) we have

$$sl_w(G_{org}) = cp = sl(G_{org}). \tag{3.4}$$

Hence, if we define the difference between $sl_w(G^R_{cls})$ and $sl_w(G_{org})$ as Δsl_w(defined in table 3.2), Δsl_w is equal to the difference between $sl_w(G^R_{cls})$ and cp (defined in table 3.2). However, as described in sec.3.3.3, the combination of tasks for each cluster depends on the task merging policy, Δsl_w can not be derived before a task clustering. Thus, we derive the upper bound of Δsl_w, i.e., $\Delta sl_{w,up}$(defined in table 3.2).

At first, let a cluster in which at least one task belongs to seq_R be $cls_R(i)$. Let assume that $cls_R(i)$ such that $cls_R(i), w(cls_R(i)) \geq \delta$ is generated. Then let the difference between "the summation of sizes of tasks which belongs to seq_R but does not belong to $seq^{\prec}_R(i)$" and "the summation of data size localized in seq_R" be ΔL_i(defined in 3.2). That is, ΔL_i can be obtained by taking the difference between "the summation of sizes of tasks without precedence relationship any one task in $seq^{\prec}_R(i)$" and "the summation of data size localized in $seq^{\prec}_R(i)$." Let the length of the subpath which consists of tasks in $seq^{\prec}_R(i)$ and communication among them in the initial state be $len(seq^{\prec}_R(i))$.

As presented in sec.3.3.3, there can be one or more combinations in $seq^{\prec}_R$. Since $seq^{\prec}_R$ is not always the critical path in G_{org}, we have

$$len(seq^{\prec}_R) \leq cp. \tag{3.5}$$

Then if we define ΔL_w as the difference between $sl_w(G^R_{cls})$ and $len(seq^{\prec}_R)$(defined in table 3.2), we have

$$\begin{aligned} \Delta sl_w &= sl_w(G^R_{cls}) - cp \\ &\leq sl_w(G^R_{cls}) - len(seq^{\prec}_R) \\ &= \Delta L_w, \\ &\Leftrightarrow \Delta sl_w \leq \Delta L_w. \end{aligned} \tag{3.6}$$

If we define the upper bound of ΔL_w as $\Delta L_{w,up}$(defined in table 3.2), $\Delta L_{w,up}$ is an upper bound of Δsl_w. Thus, we derive $\Delta L_{w,up}$ and then we define the upper bound of Δsl_w, i.e., $\Delta sl_{w,up} = \Delta L_{w,up}$.

Next, we describe the policy for deriving $\Delta L_{w,up}$. At first, the upper bound of ΔL_i is derived. By summing the upper bound for each cluster which has tasks in seq_R belong to, we derive $\Delta L_{w,up}$.

The more tasks without precedence relationship each other exist in a cluster, the larger the value of $tlevel$ of a task not included in $top_R(i)$ becomes. As a result, it can be conceivable that the upper bound of ΔL_i depends on whether each cluster in which at least one task belongs to $seq^{\prec}_R$ is linear or not. Thus, we derive the upper bound of ΔL_i for each linear cluster $cls_R(i)$ and the upper bound of ΔL_j for each non-linear cluster $cls_R(j)$, respectively. Then we derive $\Delta L_{w,up}$ by summing the upper bound of ΔL_i and the upper bound of ΔL_j for each linear cluster and each non-linear cluster.

Example 4. *At figure 3.3 (1), if we assume $s = R$, then we have*

$$\begin{aligned} \Delta L_i &= w(n^s_1) + w(n^s_2)) + w(n^s_3)) - len(seq^{\prec}_s(i)) = -(c(e^s_{1,2}) + c(e^s_{2,3})), \\ \Delta L_{i+1} &= w(n^s_4) + w(n^s_5) - len(seq^{\prec}_s(i+1)) = -c(e^s_{4,5}). \end{aligned}$$

On the other hand, at figure 3.3 (2), if we assume $s = R$, then we have

$$\begin{aligned} \Delta L_i &= \sum_{k=1}^{4} w(n^s_k) - len(seq^{\prec}_s(i)) = w(n^s_3) - (c(e^s_{1,2}) + c(e^s_{2,4})), \\ \Delta L_{i+1} &= \sum_{k=5}^{9} w(n^s_k) - len(seq^{\prec}_s(i+1)) = w(n^s_6) + w(n^s_8) - (c(e^s_{5,7}) + c(e^s_{7,9})). \end{aligned}$$

■

3.3.5 Derivation of the upper bound of the increase of WSL by generating one cluster

At first, we derive the upper bound of ΔL_i of a linear cluster and non-linear cluster, respectively as 1 and 2 described as follows.

1. The case that a cluster $cls_R(i)$, in which at least one task belongs to $seq^{\prec}_R$, is linear
   The communication among tasks in $seq^{\prec}_R(i)$ is localized by generating $cls_R(i)$,

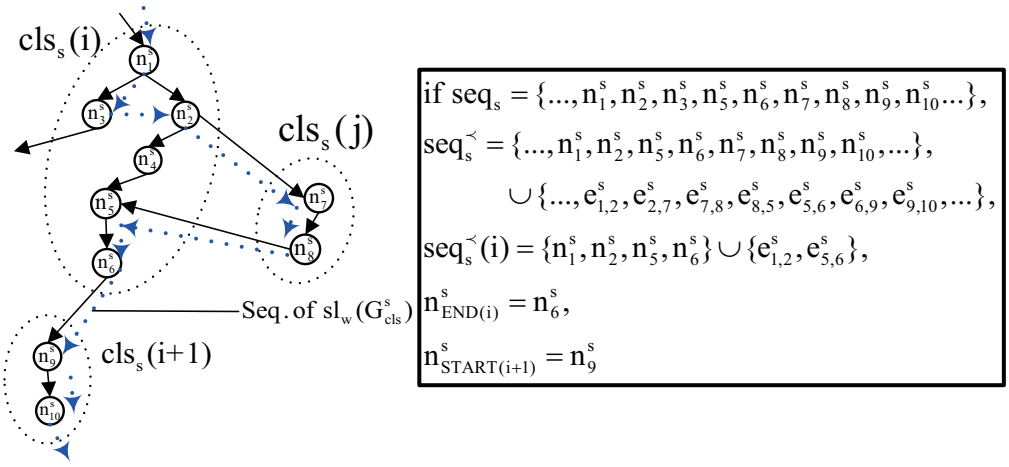


Figure 3.4: An example of $n^s_{END(i)}$ and $n^s_{START(i)}$ (appears in [22]).

$w(cls_R(i)) \geq \delta$. Here, we define

$$\begin{aligned}
&\psi_i = seq^{\prec}_R(i) \cup \{n^R_{START(i+1)} | n^R_{END(i)} \in seq^{\prec}_R(i),\\
&n^R_{END(i)} \not\prec n^R_k \ for \ \forall n^R_k \in seq^{\prec}_R(i),\\
&n^R_{START(i+1)} \in seq^{\prec}_R(i+1), n^R_{START(i+1)} \in suc(n^R_{END(i)}),\\
&e^R_{END(i),START(i+1)} \notin seq^{\prec}_R(i),\\
&e^R_{END(i),START(i+1)} \notin seq^{\prec}_R(i+1)\}.
\end{aligned} \tag{3.7}$$

That is, in eq.(3.7), $n^R_{END(i)}$ is the last executed task in $seq^{\prec}_R(i)$. Also, it means that the communication between $n^R_{END(i)}$ and $n^R_{START(i+1)}$ is not localized. For example, at figure 3.4, $n^s_{END(i)} = n^s_6$, $n^s_{START(i+1)} = n^s_9$.

ΔL_i is expressed as

$$\Delta L_i = - \sum_{e^R_{m,n} \in seq^{\prec}_{\max,S}(i)} c(e^R_{m,n}). \tag{3.8}$$

Also, if $g_{max}(n^R_k)$ in table 3.2 is applied to eq.(3.8), we have

$$\begin{aligned}
-\Delta L_i &= \sum_{e^R_{m,n} \in seq^{\prec}_R(i)} c(e^R_{m,n}) \geq \sum_{\substack{n^R_m, n^R_n \in seq^{\prec}_R(i),\\ n^R_n \in suc(n^R_m)}} \frac{w(n^R_n)}{g_{\max}(n^R_m)}\\
&= \sum_{\substack{n^R_k \in seq^{\prec}_R(i),\\ n^R_l \in \psi_i, n^R_l \in suc(n^R_k)}} \frac{w(n^R_l)}{g_{\max}(n^R_k)} - \frac{w(n^R_{START(i+1)})}{g_{\max}(n^R_{END(i)})},
\end{aligned} \tag{3.9}$$

where ψ_i is defined in eq.(3.7). Hence, the following relationship holds.

$$\Delta L_i \leq \frac{w(n^R_{START(i+1)})}{g_{\max}(n^R_{END(i)})} - \sum_{\substack{n^R_k \in seq^{\prec}_R(i), \\ n^R_l \in \psi_i, n^R_l \in suc(n^R_k)}} \frac{w(n^R_l)}{g_{\max}(n^R_k)}. \tag{3.10}$$

2. The case that a cluster $cls_R(i)$, in which at least one task belongs to $seq^{\prec}_R$, is non-linear Let the maximum cluster size be $\delta + w_{max}$(w_{max} is defined in table3.2) and we derive the upper bound of ΔL_i. In $cls_R(i)$, if every task not included in $seq^{\prec}_R(i)$ has no precedence relationship with any task included in $seq^{\prec}_R(i)$, we have

$$\begin{aligned} \Delta L_i &\leq (\delta + w_{\max}) - \sum_{n^R_k \in seq^{\prec}_R(i)} w(n^R_k) - \sum_{e^R_{p,q} \in seq^{\prec}_R(i)} c(e^R_{p,q}) \\ &\leq (\delta + w_{\max}) - \sum_{n^R_k \in seq^{\prec}_R(i)} w(n^R_k) \\ &\quad + \frac{w(n^R_{START(i+1)})}{g_{\max}(n^R_{END(i)})} - \sum_{\substack{n^R_k \in seq^{\prec}_R(i), \\ n^R_l \in \psi_i, n^R_l \in suc(n^R_k)}} \frac{w(n^R_l)}{g_{\max}(n^R_k)}. \end{aligned} \tag{3.11}$$

3.3.6 Derivation of $\Delta L_{w,up}$

To derive $\Delta L_{w,up}$, it is necessary to decide the number of clusters in each of which at least one task belongs to seq_R. $seq^{\prec}_R$ is the set of tasks having precedence relationships with each other, i.e., the set of tasks which belong to a path. Thus, at first we derive the upper bound of the number of clusters on a path. If it holds that the summation of sizes of tasks in $seq^{\prec}_R(i)$ is δ or more for any one of clusters, e.g., $cls_R(i)$, we have the following relationship with the upper bound of the number of clusters on the path $seq^R(i)$, ϕ, as ϕ_{max}.

$$\phi \leq \left\lfloor \frac{cp_w}{\delta} \right\rfloor \leq \frac{cp_w}{\delta} = \phi_{max}, \tag{3.12}$$

where cp_w is the maximum summation of sizes of tasks on the path. On the other hand, for any one of $seq^{\prec}_R(i)$, let assume that the summation of sizes of tasks in $seq^{\prec}_R(i)$ is smaller than δ. If every task which belongs to seq_R is included in a different clusters whose size are smaller than δ, only tasks which belong to the same path belong to seq_R. Thus, ϕ_{max} is the maximum number of tasks which belong to the same path.

(a) The case that the summation of sizes of tasks in $seq^{\prec}_R(i)$ is δ or more for any cluster $cls_R(i)$.
On the path, let the number of non-linear clusters be y, $\phi - y$ is the number of linear clusters on the path $seq^R(i)$. Then ΔL_w is bounded by the upper bound obtained at eq.(3.10) multiplied by $\phi - y$, plus the upper bound obtained at eq.(3.11) multiplied by y. For simplicity, if we define indices satisfying eq.(3.10) as $i = 1, 2, \ldots, \phi - y$ and

indeces satisfying eq.(3.11) as $j = \phi - y + 1, \phi - y + 2, \ldots, \phi$, we have the following result.

$$\begin{aligned}\Delta L_w \quad \leq \quad & \sum_{i=1}^{\phi-y} \left(\frac{w(n^R_{START(i+1)})}{g_{\max}(n^R_{END(i)})} \quad - \sum_{\substack{n^R_k \in seq^{\prec}_R(i), \\ n^R_l \in \psi_i, n^R_l \in suc(n^R_k)}} \frac{w(n^R_l)}{g_{\max}(n^R_k)} \right) + y(\delta + w_{\max}) \\ & -y \sum_{n^R_k \in seq^{\prec}_R(j)} w(n^R_k) \\ & + \sum_{j=\phi-y+1}^{\phi} \left(\frac{w(n^R_{START(j+1)})}{g_{\max}(n^R_{END(j)})} \quad - \sum_{\substack{n^R_k \in seq^{\prec}_R(i), \\ n^R_l \in \psi_i, n^R_l \in suc(n^R_k)}} \frac{w(n^R_l)}{g_{\max}(n^R_k)} \right) \end{aligned} \tag{3.13}$$

By developing eq.(3.13), we have

$$\begin{aligned}\Delta L_w \quad \leq \quad & \sum_{i=1}^{\phi} \frac{w(n^R_{START(i+1)})}{g_{\max}(n^R_{END(i)})} - \sum_{i=1}^{\phi} \sum_{\substack{n^R_k \in seq^{\prec}_R(i), \\ n^R_l \in \psi_i, n^R_l \in suc(n^R_k)}} \frac{w(n^R_l)}{g_{\max}(n^R_k)} + y(\delta + w_{\max}) \\ & -y \sum_{n^R_k \in seq^{\prec}_R(j)} w(n^R_k). \end{aligned} \tag{3.14}$$

At eq.(3.14), we have the following relationship.

$$\begin{aligned} \sum_{i=1}^{\phi} \frac{w(n^R_{START(i+1)})}{g_{\max}(n^R_{END(i)})} \quad & \leq \quad \phi_{\max} \max_{n^R_k \in V_S} \left\{ \frac{\max\limits_{n^R_l \in V^R_{cls}} \left\{ w(n^R_l) \right\}}{g_{\max}(n^R_k)} \right\}, \\ \min_{p \in G^0_{cls}} \left\{ \sum_{\substack{n^0_k, n^0_l \in p, \\ n^R_l \in suc(n^R_k)}} \frac{w(n^R_l)}{g_{\max}(n^R_k)} \right\} \quad & \leq \quad \sum_{i=1}^{\phi} \sum_{\substack{n^R_k \in seq^{\prec}_R(i), \\ n^R_l \in \psi_i, n^R_l \in suc(n^R_k)}} \frac{w(n^R_l)}{g_{\max}(n^R_k)}, \\ y \sum_{n^R_k \in seq^{\prec}_R(j)} w(n^R_k) \quad & \geq \quad 0. \end{aligned} \tag{3.15}$$

Thus,

$$\begin{aligned}\Delta L_w \quad \leq \quad & \phi_{\max} \max_{n^R_k \in V_S} \left\{ \frac{\max\limits_{n^R_l \in V^R_{cls}} \left\{ w(n^R_l) \right\}}{g_{\max}(n^R_k)} \right\} - \min_{p \in G^0_{cls}} \left\{ \sum_{\substack{n^0_k, n^0_l \in p, \\ n^R_l \in suc(n^R_k)}} \frac{w(n^R_l)}{g_{\max}(n^R_k)} \right\} \\ & +y(\delta + w_{\max}) = \Delta L_{w,up}. \end{aligned} \tag{3.16}$$

From eq.(3.6) and the definition of ϕ_{max} at eq.(3.12), we have

$$\Delta sl_{w,up} = \frac{cp_w}{\delta} \max_{n_k^R \in V_S} \left\{ \frac{\max_{n_l^R \in V_{cls}^R} \{w(n_l^R)\}}{g_{\max}(n_k^R)} \right\} - \min_{p \in G_{cls}^0} \left\{ \sum_{\substack{n_k^0, n_l^0 \in p, \\ n_l^R \in suc(n_k^R)}} \frac{w(n_l^R)}{g_{\max}(n_k^R)} \right\} + y(\delta + w_{\max}), \tag{3.17}$$

where the second term is the minimum of the summation of the lower bound for each data size on a path in G_{cls}^0. Thus, it does not depend on δ. With considering this fact, by differenciating eq.(3.17) with respect to δ, $\Delta sl_{w,up}$ has the local minimum (and the global minimum) value when δ takes the following value.

$$\delta = \sqrt{cp_w \max_{n_k^R \in V_S} \left\{ \frac{\max_{n_l^R \in V_{cls}^R} \{w(n_l^R)\}}{g_{\max}(n_k^R) y} \right\}} = \sqrt{cp_w \max_{n_k^0 \in V_0} \left\{ \frac{\max_{n_l^0 \in V_{cls}^0} \{w(n_l^0)\}}{g_{\max}(n_k^0) y} \right\}}. \tag{3.18}$$

From eq.(3.18) if $y > 0$, $\Delta sl_{w,up}$ has the local minimum value in terms of $\Delta sl_{w,up}$. However, if w_{max} is increased while δ is fixed, $\Delta sl_{w,up}$ is increased. ■

(b) The case that the summation of sizes of tasks in $seq_R^{\prec}(i)$ is smaller than δ. Let the number of clusters on $seq_R^{\prec}$ be ϕ and let the maximum value of the number of tasks in a path p be T_{max}. Then it follows that $\phi \leq T_{max}$. T_{max} is decided from G_{org} and does not depend on δ. Hence, in this case we have

$$\phi \leq T_{max} = \phi_{max}, \tag{3.19}$$

where ϕ is defined in table 3.2 and ϕ_{max} the upper bound of ϕ. At eq.(3.14), the following conditions are satisfied.

$$\begin{aligned}
\sum_{i=1}^{\phi} \frac{w(n_{START(i+1)}^R)}{g_{\max}(n_{END(i)}^R)} &\leq T_{max} \max_{n_k^R \in V_S} \left\{ \frac{\max_{n_l^R \in V_{cls}^R} \{w(n_l^R)\}}{g_{\max}(n_k^R)} \right\}, \\
\min_{p \in G_{cls}^0} \left\{ \sum_{\substack{n_k^0, n_l^0 \in p, \\ n_l^R \in suc(n_k^R)}} \frac{w(n_l^R)}{g_{\max}(n_k^R)} \right\} &\leq \sum_{i=1}^{\phi} \sum_{\substack{n_k^R \in seq_R^{\prec}(i), \\ n_l^R \in \psi_i, n_l^R \in suc(n_k^R)}} \frac{w(n_l^R)}{g_{\max}(n_k^R)}, \\
y \sum_{n_k^R \in seq_R^{\prec}(j)} w(n_k^R) &\geq 0
\end{aligned} \tag{3.20}$$

Hence, we have

$$\Delta sl_{w,up} = T_{max} \max_{n_k^R \in V_S} \left\{ \frac{\max_{n_l^R \in V_{cls}^R} \{w(n_l^R)\}}{g_{\max}(n_k^R)} \right\} - \min_{p \in G_{cls}^0} \left\{ \sum_{\substack{n_k^0, n_l^0 \in p, \\ n_l^R \in suc(n_k^R)}} \frac{w(n_l^R)}{g_{\max}(n_k^R)} \right\} + y(\delta + w_{\max}). \tag{3.21}$$

Thus, in this case $\Delta sl_{w,up}$ is increased with the increase of δ. ■

3.3.7 Decision of δ_{opt}

From retuls in the previous section, it is found that the behavior of $\Delta sl_{w,up}$ differs depending on whether the summation of sizes of tasks in $seq_R^{\prec}(i)$ is larger than δ or not for each cluster $cls(i)$ in which at least one task belongs to seq_R.

In sec.3.3.6 (a), $\Delta sl_{w,up}$ has the local minimum (and the global minimum) value if $y > 0$ and w_{max} is constant. However, in sec.3.3.6 (b), $\Delta sl_{w,up}$ is increased with increasing of δ. From those two characteristics, it can be concluded that merging a task without precedence relationship with a task in $seq_s^{\prec}(i)$ can lead to the increase of $\Delta sl_{w,up}$.

At eq. (3.18), if $y = 0$, it corresponds to the case every cluster in which at least one task belongs to $seq_R^{\prec}$ is linear, and then $\Delta sl_{w,up}$ is monotonically decreasing. However, how δ should be set when at least one cluster becomes non-linear is unknown before a task merging step. If $y \neq 0$, $\Delta sl_{w,up}$ is minimized when δ is equal to the right side of eq.(3.18). At the right side of eq.(3.18), if y increases, the number of non-linear clusters is increased, so that the upper bound of $sl_w(G_{cls}^R)$ is also increased. Thus, here we set $y = 1$ by assuming that the upper bound of $sl_w(G_{cls}^R)$ is minimized as much as possible. Hence, δ_{opt} is set as follow.

$$\delta_{opt} = \sqrt{cp_w \max_{n_k^0 \in V_0} \left\{ \frac{\max_{n_l^0 \in V_{cls}^0} \{w(n_l^0)\}}{g_{\max}(n_k^0)} \right\}}. \tag{3.22}$$

3.3.8 Relationship between WSL and the schedule length

In this section, we study the effect that $sl_w(G_{cls}^R)$ has on $sl(G_{cls}^R)$. At first, we denote two lemmas proved in the literature [16].

Lemma 1. $cp \leq (1 + \frac{1}{g_{min}})cp_w$.

Lemma 2. $cp_w \leq sl(G_{cls}^R)$.

Next, we describe the relationship between $\Delta sl_{w,up}$ and $sl(G_{cls}^R)$. From eq.(3.6), we have

$$\Delta sl_w = sl_w(G_{cls}^R) - cp \leq \Delta sl_{w,up}. \tag{3.23}$$

From lemma 1, eq.(3.23) becomes

$$sl_w(G_{cls}^R) - \left(1 + \frac{1}{g_{min}}\right) cp_w \leq sl_w(G_{cls}^R) - cp \leq \Delta sl_{w,up}. \tag{3.24}$$

From lemma 3, we have

$$sl_w(G_{cls}^R) - \left(1 + \frac{1}{g_{min}}\right) sl(G_{cls}^R) \leq sl_w(G_{cls}^R) - \left(1 + \frac{1}{g_{min}}\right) cp_w. \tag{3.25}$$

Hence, we obtain the following result.

$$sl_w(G^R_{cls}) - \left(1 + \frac{1}{g_{min}}\right) sl(G^R_{cls}) \leq \Delta sl_{w,up}$$
$$\Leftrightarrow sl(G^R_{cls}) \geq \frac{sl_w(G^R_{cls}) - \Delta sl_{w,up}}{1 + \frac{1}{g_{min}}}. \tag{3.26}$$

At eq.(3.17), $\Delta sl_{w,up}$ is varied with δ. Thus, by deciding δ_{opt}, $\Delta sl_{w,up}$ is fixed at eq.(3.17). From eq.(3.26), it can be said that task merging steps in order to minimize $\Delta sl_{w,up}$ after δ_{opt} has been decided can lead to minimize the lower bound of the schedule length. As defined in sec.3.3.2, the data waiting time at n^s_k is not considered for each task n^s_k such that $n^s_k \in in_R(i), n^s_k \notin top_s(i)$ in each cluster, $cls_s(i)$. It is necessary for tasks in a cluster to have precedence relationships in order to minimize the upper bound of $sl_w(G^R_{cls})$.

In this case, $sl_w(G^R_{cls})$ is the schedule length when each task is executed as early as possible without data waiting time for each task. Thus, this fact leads to the result at eq.(3.26) since the reduction of $sl_w(G^R_{cls})$ corresponds to reducing the ideal schedule length such that no data waiting time exists. Edges belonging to $seq^{\prec}_R$ have two types, i.e., localized edges and not localized edges. Moreover, edges belonging to $seq^{\prec}_R$ is not necessary on the critical path. Hence, we have the following relationship.

$$-\max_{p \in G^0_{cls}} \left\{ \sum_{\substack{n^0_k, n^0_l \in p, \\ n^0_k \in pred(n^0_l)}} c(e^0_{k,l}) \right\} \leq sl_w(G^R_{cls}) - cp. \tag{3.27}$$

The following relationship holds if and only if $sl(G^R_{cls}) \leq cp$.

$$sl_w(G^R_{cls}) - cp \leq sl_w(G^R_{cls}) - sl(G^R_{cls})$$
$$\Leftrightarrow -\max_{p \in G^0_{cls}} \left\{ \sum_{\substack{n^0_k, n^0_l \in p, \\ n^0_k \in pred(n^0_l)}} c(e^0_{k,l}) \right\} \leq sl_w(G^R_{cls}) - sl(G^R_{cls})$$
$$\Leftrightarrow sl(G^R_{cls}) \leq sl_w(G^R_{cls}) + \max_{p \in G^0_{cls}} \left\{ \sum_{\substack{n^0_k, n^0_l \in p, \\ n^0_k \in pred(n^0_l)}} c(e^0_{k,l}) \right\}, \ if \ sl(G^R_{cls}) \leq cp. \tag{3.28}$$

From eq.(3.28), it can be concluded that minimizing $sl_w(G^R_{cls})$ leads to minimize the upper bound of $sl(G^R_{cls})$, under the constraint that $sl_w(G^R_{cls})$ is smaller than the critical path length.

3.4 Task clustering algorithm

3.4.1 Requirements for the algorithm

In sec.3.3, we derived the lower bound of every cluster size when $sl_w(G^R_{cls})$ is minimized. Furthermore, in the section, we proved that the schedule length can be minimized by merg-

INPUT: G_{org}
OUTPUT: G^R_{cls}
Define UEX_s as a set of clusters which satisfy eq.(3.29).;
Define RDY_s as a set of clusters which statisfy eq.(3.31).;
For each $n_k \in V$, let $n^0_k \leftarrow n_k, cls_0(k) = \{n^0_k\}$ and put $cls_0(k)$ into V^0_{cls}.
0. Derive δ_{opt} by eq.(3.22).
1. $E_0 \leftarrow E, UEX_0 \leftarrow V^0_{cls}, RDY_0 \leftarrow \{cls_0(k) | cls_0(k) = \{n^0_k\}, pred(n^0_k) = \emptyset\}$;
2. **WHILE** $UEX_s \neq \emptyset$ **DO**
3. $pivot_s \leftarrow getPivot(RDY_s)$; /* detailed in Sec.3.4.5. */
4. $target_s \leftarrow getTarget(pivot_s)$; /* Detailed in Sec.3.4.6. */
5. $RDY_{s+1} \leftarrow clustering(pivot_s, target_s)$; /* Detailed in Sec.3.4.7. */
6. **ENDWHILE**
7. **RETURN** G^R_{cls};

Figure 3.5: Whole procedures of our proposing task clustering (appears in [22]).

ing tasks by which $sl_w(G^R_{cls})$ is minimized. Thus, the following points are needed for implementing a task clustering.

(i) Perform task merging steps until each cluster size is δ_{opt} or more.
By performing task merging steps so that each cluster size is δ_{opt} or more defined at eq.(3.22), at least a set of clusters, by which the upper bound of $sl_w(G^R_{cls})$ is minimized, is obtained.

(ii) For each task in $seq^{\prec}_s$, merge them with precedence relationships.
From the result obtained in sec.3.3.6 (a), it is necessary to merge tasks which have precedence relationships with a task in $seq^{\prec}_s(i)$ for each cluster $cls_s(i)$ such that the size of $cls_s(i))$ is smaller than δ in order to minimize the upper bound of $sl_w(G^R_{cls})$.

(iii) Minimize $sl_w(G^s_{cls})$ as much as possible by each task merging step.
From the result obtained in sec.3.3.8, minimizing $sl_w(G^R_{cls})$ by task merging steps can lead to the minimization of the lower bound of the schedule length. Moreover, by satisfying the requirement (ii), the upper bound of $sl_w(G^R_{cls})$ can be minimized, but the actual increase or decrease of $sl_w(G^R_{cls})$ is unknown before task merging steps. Thus, during task merging steps, just like (ii), a task in $seq^{\prec}_s$ should be selected as a candidate for a task merging step, and then another task, by which $sl_w(G^{s+1}_{cls})$ can be minimized, should be selected for the task merging step.

3.4.2 Summary of the algorithm

Figure 3.5 shows the summary of the proposed task clustering algorithm. At the initial stage, the initial DAG is put into G^0_{cls}, i.e., $G^0_{cls} \leftarrow G_{org}$. Then let each task in V^0_{cls} belong to a cluster having only one task. That is, for $cls_0(k) \in V^0_{cls}$, let $cls_0(k) = \{n^0_k\}$, where we assume that $\{cls_0(1) = \{n^0_1\}, cls_0(2) = \{n^0_2\}, \dots\}$. It is necessary to specify which cluster size is δ_{opt} or more in order to satisfy the requirement sec.3.4.1 (i). Hence, we define UEX_s, which contains the set of clusters satisfying the following condition.

$$\{cls_s(i) | w(cls_s(i)) < \delta_{opt}\}. \tag{3.29}$$

We assume that each cluster in V_{cls}^s(where each cluster has only one task) belongs to UEX_s before a task merging step.

The main procedures of the algorithm correspond to the range from line 2 - 6 in figure.3.5. From sec.3.4.1 (ii) and (iii), one of policies of the task clustering algorithm is to select two clusters such that at least one task in them belong to $seq_s^{\prec}$ and both sizes are smaller than δ_{opt} in order to reduce $sl_w(G_{cls}^{s+1})$ by a task merging step, i.e., $sl_w(G_{cls}^s) > sl_w(G_{cls}^{s+1})$. Then the algorithm try to make each cluster size larger by merging those two tasks into a cluster.

Here let the cluster selected according to a certain priority from the set of clusters expressed as RDY_s (defined in sec.3.4.4) be $pivot_s$ (line 3 in figure 3.5). Then let the cluster which has precedence relationships with at least one task in $pivot_s$ be $target_s$ (defined in sec.3.4.6) be $target_s$. The algorithm merges $pivot_s$ and $target_s$ into a cluster (line 5 in figure 3.5). After such a task merging step, every task in $target_s$ comes to be included in $pivot_s$, so that they are not included in UEX_{s+1} and RDY_{s+1} (if $target_s \in RDY_s$). At line 5 in figure 3.5, $pivot_s$ is removed from both UEX_{s+1} and RDY_{s+1} if the sum of cluster size among $pivot_s$ and $target_s$ is δ_{opt} or more. At the $(s+1)$-th task merging step, update procedures of the selecting priority for $pivot_{s+1}$ and procedures of adding clusters into RDY_{s+1} at line 5 in figure 3.5 are described in sec.3.4.7.

When every cluster size is δ_{opt}, i.e., $UEX_S = \emptyset$, the algorithm is finished. It is necessary to decide following points to minimize $sl_w(G_{cls}^R)$ by the algorithm.

1. The policy for task merging steps with considering the time complexity.
2. Criteria for selecting $pivot_s$ (at line 3 in figure 3.5).
3. Criteria for selecting $target_s$ (at line 4 in figure 3.5).
4. Update policy for the selection priority of $pivot_s$ (at line 5 in figure 3.5).

3.4.3 Policy for task merging steps

After a task in seq_s and a task having a precedence relationship with it are merged, at most $sl_w(G_{cls}^{s+1})$ is reduced by communication data size, i.e., $sl_w(G_{cls}^s) > sl_w(G_{cls}^{s+1})$. If tasks having no precedence relationship each other are included in the same cluster, $sl_w(G_{cls}^{s+1})$ may be increased, i.e., $sl_w(G_{cls}^s) < sl_w(G_{cls}^{s+1})$. Thus, if the cluster in which at least one task in seq_s is used for the task merging step as $pivot_s$, $sl_w(G_{cls}^{s+1})$ may be increased or decreased.

Here, before the $(s+1)$-th task merging step, let assume the polilcy to select the cluster as $target_s$ which has the maximum LV_s value in clusters each of which has at least one task in seq_s, and by which $sl_w(G_{cls}^{s+1})$ can be minimized. In this case, it occurs two problems when each cluster's LV_s is updated to select one task in seq_s.

1. Problems in terms of the time complexity
   After the $(s-1)$-th task merging steps between $pivot_{s-1}$ and $target_{s-1}$, there is no way to know the set of seq_s without tracing every task in the DAG. It takes $|V_s| + |E_s|$ steps to update both $tlevel$ value and $blevel$ for each task in V_s. Then it takes $\log |top_s(i)|$ steps to derive $TL_s(i)$ and $BL_s(i)$ for a cluster $cls_s(i)$. Thus, by updating LV_s value for every cluster in V_{cls}^s it takes $|V_{cls}^s|(\log |top_s(i)| + \log |out_s(i)|)$ steps. Those procedures

are repeated until every cluster size is δ_{opt} or more. As a result, the time complexity taken by updating LV_s value for each cluster in the DAG is

$$O_{all} \quad = \quad O(|V|(|V| + |E|) + |V|^2 \log |V|), \tag{3.30}$$

even if $pivot_s$ is removed from UEX_s by only one task merging step.

In general, one point for reducing the time complexity in a task clustering is how to reduce that taken by updating the task selection priority. The priority in conventional task clustering is the schedule length. One algorithm requires the time complexity of $O(|E|(|V| + |E|))$ for updating the schedule length by the full tracing [27], and the another one requires the time complexity of $O(|E| \log |V|)$ or $O(|V| \log |V|)$ by limiting the tracing area in order to reduce the time complexity. On the other hand, there is one algorithm which does not perform updating the schedule length [29].

The time complexity of O_{all} taken to updating LV_s by the full tracing can become a critical problem in terms of the time complexity, because it is higher than $O(|E|(|V| + |E|))$ required to updating the schedule length by the full tracing.

2. It does not always lead to the decrease of $sl_w(G^s_{cls})$
   When size of every cluster in which at least one task belongs to seq_s is δ_{opt} or more, other cluster's sizes may be smaller than δ_{opt}. In such a case, it is necessary to make the cluster in which every task does not belong to seq_s $pivot_s$ in order to satisfy sec.3.4.1 (i). As a result, seq_{s+1} is updated by the task merging step between the selected $pivot_s$ and $target_s$, and then $sl_w(G^{s+1}_{cls})$ can be increased, i.e., $sl_w(G^s_{cls}) < sl_w(G^{s+1}_{cls})$. That is, even if a cluster in which at least one task belongs to seq_s is selected as $pivot_s$ for every task merging step, there is no guarantee that $sl_w(G^{s+1}_{cls})$ is decreased, i.e., $sl_w(G^s_{cls}) > sl_w(G^{s+1}_{cls})$.

From problems described above, the policy in our proposal is to select $pivot_s$ in a certain set of clusters without specifying and updating seq_s. Then the proposal updates LV_{s+1} for each cluster in the set of clusters after the s-th task merging step. Even if every cluster in which at least one task in seq_s has precedence relationships with each other, it is possible that a task included in seq_s and a task not included in seq_s are merged in a cluster, and then tasks without precedence relationships are included in the cluster. That is, repeating task merging steps until every cluster size is δ_{opt} or more leads to the increase of $sl_w(G^{s+1}_{cls})$, i.e., $sl_w(G^s_{cls}) < sl_w(G^{s+1}_{cls})$. Thus, the point in terms of performance of the algorithm is that how to reduce $sl_w(G^{s+1}_{cls})$, i.e., $sl_w(G^s_{cls}) > sl_w(G^{s+1}_{cls})$ by selecting $pivot_s$ and $target_s$, provided that both seq_s and $sl_w(G^s_{cls})$ are not specified for every task merging steps.

3.4.4 Definition of the range for selecting $pivot_s$ and its effect

In the light of facts described above, the algorithm select $pivot_s$ from the set of cluster satisfying the following condition.

$$\begin{aligned} &\{cls_s(r) | cls_s(r) \in UEX_s, pred(n^s_{r'}) = \emptyset, cls_s(r) = \{n^s_{r'}\}\} \\ &\cup \left\{ \begin{array}{l} cls_s(r) | cls_s(r) \in UEX_s, cls_s(q) \notin UEX_s, \\ n^s_{q'} \in cls_s(q), n^s_{q'} \in pred(n^s_{r'}) \; for \; \forall n^s_{r'} \in top_s(r) \end{array} \right\}. \end{aligned} \tag{3.31}$$

In RDY_s, the set of clusters satisfying the condition of eq.(3.31) is retained. That is, RDY_s at line 3 in figure 3.5 is the set of clusters included in UEX_s and at the same time in which only one START task is included, or the set of clusters in each (let $cls_s(i)$) of which immediate predecessor tasks of any task in $top_s(i)$ belong to a cluster whose size is δ_{opt} or more.

Before the $(s+1)$-th task merging step, at line 3 in figure 3.5 size of every cluster in RDY_{s+1} is smaller than δ_{opt} by the update procedure from RDY_s to RDY_{s+1} at line 5 (detailed in sec.3.4.7). That is, the algorithm makes only the set of clusters that are indexed as $s \leftarrow s+1$ belong to RDY_{s+1} by the procedure at line 5 in figure 3.5. At line 1 in figure 3.5, all clusters are added into UEX_s. At this time, the set of clusters in each of which only START task is included is added into RDY_s(At line 1 in figure 3.5, the number of tasks in each cluster is 1).

Next we describe how the policy to select $pivot_s$ from clusters in RDY_s effects on the behavior of LV_{s+1} for each cluster. At first, assume that a cluster $cls_s(r)$ in RDY_s is selected as $pivot_s$. Then according to sec.3.4.1 (ii), let a cluster selected from the set of clusters $cls_s(t)$ defined as follow be $target_s$ (As for the detailed selection policy for $target_s$, see sec.3.4.6).

$$TGT_s(r) = \{cls_s(t) | n_{t'} \in suc(n_{r'}), cls_s(t) = \{n_{t'}\}, cls_s(t) \in UEX_s, \exists n_{r'} \in out_s(r)\}. \tag{3.32}$$

As a result, the following theorem holds.

Thorem 3.1. *If $cls_{s+1}(r) \leftarrow cls_s(r) \cup cls_s(t)$, then we have $TL_{s+1}(r) = TL_s(r)$.*

Proof 1. *It must be satisfied that $n^s_{t'} \prec n^s_{r'}$ for each task $n^s_{r'}$ in $top_s(r)$ with task $n^s_{t'}$ in $cls_s(t)$ in order to satisfy $TL_{s+1}(r) - TL_s(r) \neq 0$. Thus, we prove the theorem in two cases.*

Case 1: The case of $n^s_{t'} \in pred(n^s_{r'})$.
From eq.(3.31), any immediate predecessor task of $top_s(r)$ belongs to the cluster whose size is δ_{opt} or more, but this assumption conflicts with $w(cls_s(t)) < \delta_{opt}$.

Case 2: The case that it is assumed that $n^s_{t'} \notin pred(n^s_r),\ n^s_{t'} \prec n^s_{r'}$
For a cluster $cls_s(l)$, it is assumed that there is one immediate predecessor task of $top_s(r)$ in $cls_s(l)$. Then from the definition of RDY_s, it can be said that $w(cls_s(l)) \geq \delta_{opt}$. It is necessary that $n^s_{t'}$ is an immediate predecessor task of a task in $top_s(l)$, or $n^s_{t'}$ must be executed before a task in $top_s(l)$ in order to to satisfy $TL_{s+1}(r) - TL_s(r) \neq 0$.

In the former case, $cls_s(l)$ does not belong to UEX_s, and size of the cluster in which a task is one of immediate predecessor tasks of $top_s(l)$ is δ_{opt} or more. Hence, similar to case 1, this assumption conflicts with $w(cls_s(t)) < \delta_{opt}$. Moreover, in the latter case, the confliction can be derived by repeating case 2 until $n^s_{t'}$ is one of immediate predecessor tasks of $top_s(l)$.

From both cases, the theorem is proved. ■

Also, the following theorem 3.2 is proved by the same manner as 3.1.

Thorem 3.2. *Let $cls_s(r), cls_s(s) \in RDY_s$. If we assume that $pivot_s$ is $cls_s(r)$ and a cluster $cls_s(t)$ in $TGT_s(r)$ is merged as $target_s$, we obtain $TL_{s+1}(s) = TL_s(s)$.*

Proof 2. *For a task $n^s_{r'}$ in $cls_s(r)$ with a task $n^s_{s'}$ in $top_s(s)$, if it holds $n^s_{r'} \prec n^s_{s'}$, it is assumed that $TL_{s+1}(s) - TL_s(s) \neq 0$ is possible. Thus, it is enough to prove the conflict of this assumption by considering two cases as follows.*

Case 1: The case that we assume $n^s_{r'} \in pred(n^s_{s'})$
Before the task merging step, both $cls_s(r)$ and $cls_s(t)$ belong to UEX_s. Thus, it conflicts with $cls_s(s) \in RDY_s$.

Case 2: The case that it is assumed that $n^s_{r'} \notin pred(n^s_{s'}),\ n^s_{r'} \prec n^s_{s'}$
For a cluster $cls_s(l)$, assume that a task in $cls_s(l)$ is an immediate predecessor task of a task in $top_s(s)$. Then we have $w(cls_s(l)) \geq \delta_{opt}$. To satisfy $TL_{s+1}(s) - TL_s(s) \neq 0$, $n^s_{r'}$ must be an immediate predecessor task of $top_s(l)$, or $n^s_{r'}$ must be executed before a task in $top_s(l)$.

In the former case, $cls_s(l)$ does not belong to UEX_s, and size of the cluster in which any immediate predecessor task of $top_s(l)$ is δ_{opt} or more. Threfore, size of both $cls_s(r)$ and $target_s$ δ_{opt} or more, which conflicts with the assumption. In the latter case, by repeating case 2 until $n^s_{r'}$ is an immediate predecessor task of $top_s(l)$, the conflict is derived. ■

From theorem 3.1 and 3.2, for each cluster in RDY_s, TL_{s+1} equals to TL_s, even if the algorithm performs a task merging step with selecting $cls_s(r)$ in RDY_s as $pivot_s$ and selecting the cluster $cls_s(t)$ in $TGT_s(r)$ as $target_s$. Thus, by doing such a task merging step, it is not necessary to update TL_{s+1} values for both a cluster in RDY_s and a cluster whose size is δ_{opt} or more. The policy to select $pivot_s$ from RDY_s can have an advantage than the policy that updating LV_{s+1} by a full tracing in terms of the time complexity.

Next, we describe how the policy that selects $pivot_s$ from RDY_s and select $target_s$ based on eq.(3.32) has effect on $sl_w(G^{s+1}_{cls})$. Obviously, at line 1 in figure 3.5, initially every START task belongs to RDY_0. At that moment, at least one START task belongs to seq_0, since every task in seq_0 belongs to the critical path. However, it is not always that at least one task in seq_s belongs to a cluster in RDY_s after s task merging steps. Threfore, it is necessary to study how LV_{s+1} value is varied for each cluster in which at least a task seq_{s+1} by a task merging step in order to clarify the increase and decrease of $sl_w(G^{s+1}_{cls}) - sl_w(G^s_{cls})$ after the $(s+1)$-th task merging step. Here, the following theorem holds by defining a cluster as $cls_s(i)$ in which at least one task belongs to seq_s.

Thorem 3.3. *For a cluster $cls_s(i)$ in which at least one task belongs to seq_s, let $cls_s(i) = \{n^s_{i'}\}$ and $cls_s(i) \notin RDY_s$. Let the cluster $cls_s(r)$ selected from RDY_s be $pivot_s$, and let the cluster $cls_s(t)$ included in $TGT_s(r)$ be $target_s$, where any task in $pivot_s$ and $target_s$ does not belong to seq_s. Here, if we assume $\Delta TL_{s+1} = TL_{s+1}(i) - TL_s(i), \Delta BL_{s+1} = BL_{s+1}(i) - BL_s(i)$, then the following relationships hold.*

(A). $n^s_{r'} \prec n^s_{i'}\ for\ \exists n^s_{r'} \in cls_s(r) \cup cls_s(t) \Rightarrow \Delta TL_{s+1}(i) \geq 0, \Delta BL_{s+1}(i) = 0.$

(B). $n^s_{i'} \prec n^s_{r'}\ for\ \exists n^s_{r'} \in cls_s(r) \cup cls_s(t) \Rightarrow \Delta TL_{s+1}(i) = 0, \Delta BL_{s+1}(i) = 0.$

Proof 3. *At first, we prove (A). Since only one task belongs to $cls_s(i)$, abviously $TL_s(i) = tlevel(n^s_{i'})$, $BL_s(i) = w(n^s_{i'}) + blevel(n^s_{i'})$. If tasks without precedence relationships in $cls_s(r)$ and $cls_s(t)$ are included in $cls_{s+1}(r)$ by a task merging step, $tlevel(n^{s+1}_{r'})$ can be increased, i.e.,*

$$tlevel(n^s_{r'}) < tlevel(n^{s+1}_{r'}).$$

Thus, $tlevel(n_{i'}^{s+1})$ is also increased, i.e., $tlevel(n_{i'}^{s}) < tlevel(n_{i'}^{s+1})$. On the other hand, if $cls_{s+1}(r)$ is linear after the task merging step, every task in $cls_{s+1}(r)$ has precedence relationships. Thus, $TL_{s+1}(i)$ is not increased, i.e., $TL_s(i) \geq TL_{s+1}(i)$. Also, $BL_{s+1}(i)$ is not varied, i.e., $BL_s(i)$ is equal to $BL_{s+1}(i)$ because $n_{r'}^{s} \prec n_{i'}^{s}$.

Next we prove (B). Since $n_{i'}^{s} \prec n_{r'}^{s}$, $tlevel(n_{i'}^{s+1})$ is not varied by a task merging step. In $cls_{s+1}(r)$, the localized data between $cls_s(r)$ and $cls_s(t)$ has effect on $blevel$ value for each task in $cls_{s+1}(r)$. Thus, $blevel$ value for each task $cls_{s+1}(r)$ is not varied. Also, both $cls_s(r)$ and $cls_s(t)$ do not have a task which belongs to seq_s, and then the variation of $blevel$ value for each task in those two clusters does not effect on the variation of $blevel(n_{i'}^{s+1})$. Hence, the theorem is proved.■

Thorem 3.4. *For each cluster in which at least one task belongs to seq_s, let $|cls_s(i)| \geq 2$. Also, let the cluster $cls_s(r)$ selected from RDY_s be $pivot_s$, let the cluster $cls_s(t)$ in $TGT_s(r)$ be $target_s$, and assume that any task in $pivot_s$ and $target_s$ does not belong to seq_s. At that moment, if the increase and decrease of $TL_{s+1}(i)$ and $BL_{s+1}(i)$, are defined as $\Delta TL_{s+1}(i)$ and $\Delta BL_{s+1}(i)$ by the task merging of $cls_{s+1}(r) \leftarrow cls_s(r) \cup cls_s(k)$, for $\exists n_{i'}^{s} \in cls_s(i)$ the following relationships hold.*

(A). $n_{r'}^{s} \prec n_{i'}^{s}$ *for* $\exists n_{r'}^{s} \in cls_s(r) \cup cls_s(t) \Rightarrow \Delta TL_{s+1}(i) = 0,\ \Delta BL_{s+1}(i) = 0.$

(B). $n_{i'}^{s} \prec n_{r'}^{s}$ *for* $\exists n_{r'}^{s} \in cls_s(r) \cup cls_s(t) \Rightarrow \Delta TL_{s+1}(i) = 0,\ \Delta BL_{s+1}(i) = 0.$

Proof 4. *Since the number of tasks in $cls_s(i)$ is two or more, $cls_s(i)$ belongs to RDY_s or its size is δ_{opt} or more. Thus, any immediate predecessor task of $top_s(i)$ also belongs to a cluster whose size is δ_{opt} or more. Here, it hold that $\Delta TL_{s+1}(i) = 0$ in both case (A) and (B) from theorem 3.2.*

Next, we prove $\Delta BL_{s+1}(i) = 0$ at (A). Though the number of tasks in $cls_s(i)$ is two, $\Delta BL_{s+1}(i)$ is equal to 0, which is similar to theorem 3.3 because $n_{r'}^{s} \prec n_{i'}^{s}$.

Finally, we prove $\Delta BL_{s+1}(i) = 0$ at (B). Similar to theorem 3.3 (B), $cls_s(r)$ and $cls_s(t)$ do not have any task in seq_s. Thus, the variation of $blevel$ values of tasks in those two clusters do not have effect on $blevel$ value for each task in $cls_{s+1}(i)$. ■

From theorem 3.3 and 3.4, it can be said that the policy that selecting the cluster $cls_s(r)$ from RDY_s as $pivot_s$ and selecting the cluster $cls_s(t)$ from $TGT_s(r)$ as $target_s$ can lead to the increase of TL_{s+1} value of a cluster to which only one task belongs, i.e., $TL_s < TL_{s+1}$. On the other hand, the policy does not have effect on LV_{s+1} value for each cluster having two or more tasks. Also, for any cluster $cls_s(i)$ in which at least one task belongs to seq_s the policy does not have effect on its $BL_{s+1}(i)$ value.

Example 5. *Figure 3.6 shows an example of theorem 3.4. In this figure, a dashed line means a precedence relationship, while a dashed area means that tasks in the area belong to seq_s. Assume that a task in $cls_s(1)$ belong to seq_s, and $cls_s(4)$, $cls_s(6)$, $cls_s(7)$ and $cls_s(9)$ belong to UEX_s. Here, we assume two task merging steps, i.e., the one is that $pivot_s$ is $cls_s(4)$ and $target_s$ is $cls_s(6)$, and the other is that $pivot_{s+1}$ is $cls_{s+1}(7)$ and $target_{s+1}$ is $cls_{s+1}(9)$.*

(b) means the state after those two task merging steps have been performed. In this state, size of the cluster to which the immediate predecessor task of $top_s(1)$ (i.e., n_1^s) belongs is δ_{opt} or more, since $cls_s(1)$ has been generated before the state of (a). Thus, any task in both $cls_s(4)$ and $cls_s(6)$ is not an immediate predecessor task of n_1^s. As a result, the variation of $tlevel$ value of every task in $cls_{s+2}(4)$ at figure 3.6 (b) does not have effect on $TL_{s+2}(1)$. On the other hand, after the task merging step between $cls_s(4)$ and $cls_s(6)$ at figure 3.6 (a), at (b) we have $BL_{s+2}(7) - BL_s(7) \neq 0$

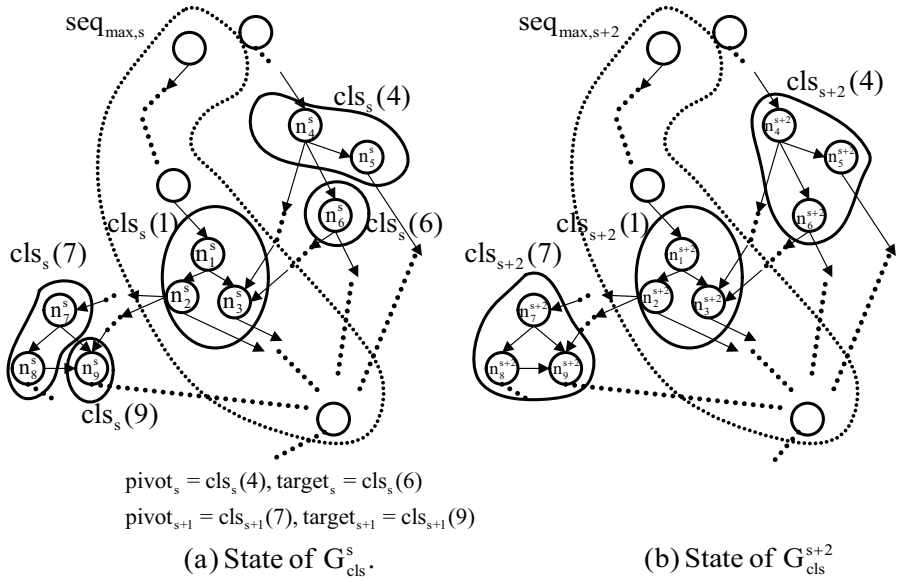


Figure 3.6: Effect on LV_{s+1} value of every cluster after one task merging step of $pivot_s$ and $target_s$
(appears in [22]).

by the task merging step between $cls_{s+1}(7)$ and $cls_{s+1}(9)$. However, blevel value of any task in $cls_{s+2}(7)$ is decreased but not increased by the data localization with compared to the state after the s-th task merging step. As a result, any task in $cls_{s+2}(7)$ at figure 3.6 (b) does not belong to seq_{s+2}, and Hence we have $BL_{s+2}(1) = BL_{s+1}(1) = BL_s(1)$. ■

3.4.5 Selection of $pivot_s$

In this section we describe $getPivot(RDY_s)$ at line 3 in figure 3.5. If we define the cluster in which at least one task belongs to seq_s as $cls_s(i)$, it is possible that $cls_s(i)$ does not belong to RDY_s. If $cls_s(i)$ belong to RDY_s, its LV_s value takes the maximum in RDY_s. In such a case, to make $LV_{s+1}(i)$ decreased (i.e., $LV_s(i) > LV_{s+1}(i)$) by the task merging step with $cls_s(i)$ being $pivot_s$ leads to the decrease of $sl_w(G^{s+1}_{cls})$ (i.e., $sl_w(G^s_{cls}) > sl_w(G^{s+1}_{cls})$). However, if $cls_s(i)$ does not belong to RDY_s, from theorem 3.3 and 3.4 the task merging step with the cluster selected from RDY_s being $pivot_s$ and the cluster selected from $TGT_s(r)$ being $target_s$ can lead to the increase of $TL_{s+1}(i)$, i.e., $TL_s(i) < TL_{s+1}(i)$. Here, let the cluster having the maximum LV_s vlaue in RDY_s, and let $cls_s(r)$ be $pivot_s$. If the cluster $cls_s(t)$ is selected from $TGT_s(r)$ as $target_s$, from theorem 3.3 and 3.4 the following theorem holds.

Thorem 3.5. *Assume that $cls_s(r) \in RDY_s$ and the cluster in seq_s is $cls_s(i)$. Any task in $cls_s(r)$ does not belong to seq_s, while $cls_s(r)$ is $pivot_s$ and the cluster selected from $TGT_s(r)$ is $target_s$. If $cls_s(r)$ is linear, and $cls_{s+1}(r)$ is linear after those clusters have been merged, from theorem 3.3 and 3.4 $TL_{s+1}(i)$ is not increased, i.e., $TL_s(i) \geq TL_{s+1}(i)$. Also, $BL_{s+1}(i)$ is not varied, i.e., $BL_s(i) = BL_{s+1}(i)$.*

Thorem 3.6. *Assume $cls_s(r) \in RDY_s$. If at least one task in $cls_s(r)$ belongs to seq_s, the increase or decrease of $LV_{s+1}(r)$ by a task merging step with $cls_s(r)$ being $pivot_s$, $sl_w(G_{cls}^{s+1})$ is also increased or decreased.*

The reason that theorem 3.5 holds is that both $TL_s(r)$ and $BL_s(r)$ are unchanged after a task merging step has been performed, due to the fact that $cls_s(r)$ is linear. Also, from theorem 3.6, it is necessary to select the cluster having the maximum LV_s value in RDY_s as $pivot_s$ to decrease $sl_w(G_{cls}^{s+1})$, i.e., $sl_w(G_{cls}^{s}) > sl_w(G_{cls}^{s+1})$. Hence, we define the cluster satisfying the following condition as $pivot_s$.

$$cls_s(p) \in RDY_s, LV_s(p) = \max_{cls_s(r) \in RDY_s} \{LV_s(r)\} \tag{3.33}$$

3.4.6 Selection of $target_s$

In this section, we describe the procedure of $getTarget(pivot_s)$ at line 4 in figure 3.5.

Condition for selecting $target_s$

If any task in $pivot_s$ does not belong to seq_s, from theorem 3.5 it is necessary to merge only tasks which have precedece relationships with any task in $pivot_s$. On the other hand, if $pivot_s$ has a task which belongs to seq_s, from theorem 3.1 and 3.2 it is necessary to select the cluster as $target_s$ by which BL_{s+1} value of $pivot_s$ is decreased, i.e., $BL_s > BL_{s+1}$ or minimize the increase by a task merging step. Also, from theorem 3.5, it is necessary to use the policy such that it maintains a cluster remains linear to decrease or minimize the increase of LV_{s+1} for each cluster. Thus, as for a task merging step with a linear cluster being $pivot_s$, we prove the following theorem.

Thorem 3.7. *Let $pivot_s$ by eq.(3.33) be $cls_s(p)$. Also, for a task $n_{t'}^s$, let $n_{t'}^s \in suc(n_{p'}^s), n_{p'}^s \in btm_s(p)$, $cls_s(t) = \{n_{t'}^s\}$. If $cls_s(p)$ is linear, $cls_{s+1}(p)$ is still linear after $cls_{s+1}(p) \leftarrow cls_s(p) \cup cls_s(t)$.*

Proof 5. *Let consider that the case $cls_{s+1}(p)$ becomes non-linear after a task merging step. In such a case, it is true that n_k^s such that $n_{t'}^s \not\prec n_k^s$ and $n_k^s \not\prec n_{t'}^s$, $n_k^s \in cls_s(p)$ exists. However, $cls_s(p)$ before a task merging step, any task in $cls_s(p)$ can be executed before $n_{p'}^s$. Thus, we have $n_k^s \prec n_{p'}^s$. As a result, we have $n_k^s \prec n_{p'}^s \prec n_{t'}^s$, i.e., $n_k^s \prec n_{t'}^s$, which has the conflict.* ■

Next, at the case that both TL_{s+1} value and BL_{s+1} value are increased, i.e., $TL_s < TL_{s+1}$, $BL_s < BL_{s+1}$ for the cluster which has been $pivot_s$, the following theorem holds.

Thorem 3.8. *Let $pivot_s$ selected by eq.(3.33) be $cls_s(p)$. Also, let the cluster $cls_s(t)$ to which an immediate successor task of $out_s(p)$ belongs and whose size is smaller than δ_{opt} be $target_s$, i.e., $target_s \neq pivot_s$. Then if $|cls_s(t)| \geq 2$, we have $n_{p'}^s \notin suc(n_{t'}^s)$ and $n_{p'}^s \notin pred(n_{t'}^s)$ for $n_{p'}^s, n_{t'}^s$ such that $\forall n_{p'}^s \in top_s(p), \forall n_{t'}^s \in top_s(t)$.*

Proof 6. *It is true that $cls_s(t) \in RDY_s$ by the fact that $|cls_s(t)| \geq 2$ and $w(cls_s(t)) < \delta_{opt}$. Thus, in this case two cluster in RDY_s are merged.*

At first, assume $n_{p'}^s \in suc(n_{t'}^s)$. However, it conflicts with $cls_s(p) \in RDY_s$ because we have $cls_s(t) \in RDY_s$. Next, assume $n_{p'}^s \in pred(n_{t'}^s)$. However, it conflects with $cls_s(t) \in RDY_s$ because $cls_s(p) \in RDY_s$.

From those conflicts, the theorem is proved.■

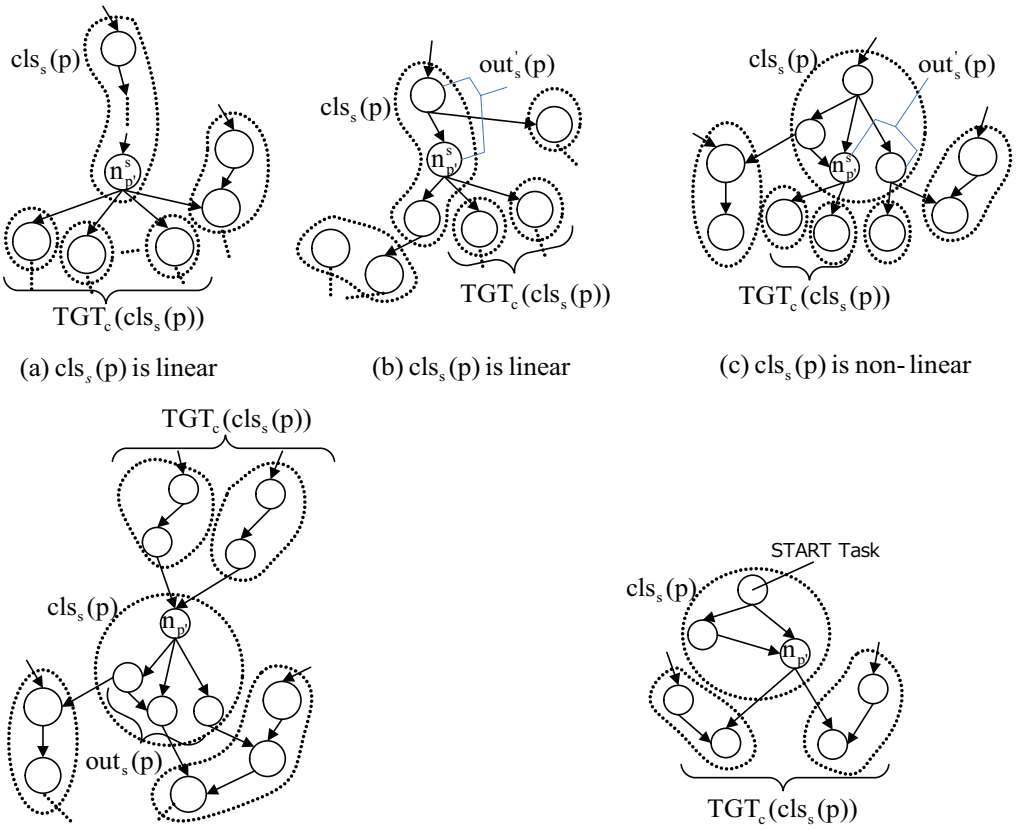


Figure 3.7: Pattern of selecting $target_s$(where let $pivot_s = cls_s(p)$) (appears in [22]).

From theorem 3.8, the number of tasks in $top_{s+1}(p)$ is increased when the cluster which has two or more tasks is selected as $target_s$ and $pivot_s$ is $cls_s(p)$. That is, this fact means that tasks without precedence relationships are included in the same cluster. As a result, not only $tlevel$ value for each task not included in $top_{s+1}(p)$ in $cls_{s+1}(p)$, but also both $TL_{s+1}(p)$ value and $BL_{s+1}(p)$, i.e., $LV_{s+1}(p)$ are increased, i.e., $BL_s(p) < BL_{s+1}(p)$. On the other hand, like theorem 3.1, if the cluster having only one task is selected as $target_s$, TL_{s+1} of the cluster which remains $pivot_s$ after a task merging step is not increased, i.e., $TL_s \geq TL_{s+1}$. Thus, it is necessary to select the cluster satisfying the following condition as $target_s$.

Cond. 1. The cluster which makes BL_{s+1} value of the cluster which has been $pivot_s$ decreased, i.e., $BL_s > BL_{s+1}$, or which minimizes the increase of BL_{s+1} value (from theorem 3.1, 3.2, and 3.8).

Cond. 2. The cluster which has precedence relationships with any task in $pivot_s$ (from theorem 3.5, 3.7, and 3.8).

Selection of $target_s$

Based on conditions described in the previous section, in this section we present the procedure for selecting $target_s$. In the procedure at line 4 in figure 3.5, at first it search for the set of clusters whose tasks have precedence relationships with $pivot_s$, and then in such clusters it select the one by which $sl_w(G_{cls}^{s+1})$ is decreased (or its increase is minimized) is selected as $target_s$. The criterion for selecting $target_s$ depends on the state of each immediate successor task of $pivot_s$ and the state of $pivot_s$ itself (i.e., linear or non-linear). Figure 3.7 shows all patters of selecting $target_s$. In figure 3.7, let $pivot_s$ be $cls_s(p)$, and let the set of candidate clusters for $target_s$ be $TGT_c(cls_s(p))$. Then the procedure means which cluster is selected as $target_s$ from $TGT_c(cls_s(p))$. In the figure, if the selection policies for $target_s$ among the cases of linear and non-linear are the same irrespective of whether $cls_s(p)$ is linear or not, only the one case is shown. At line 4 in figure the order of the search for $target_s$ is (a), (b), (c), (d), and (e). For example, if $TGT_c(cls_s(p))$ is not found at (a), then the algorithm tries to find $TGT_c(cls_s(p))$ in (b), otherwise the procedure goes to (c) ... . When $TGT_c(cls_s(p))$ is found, the algorithm select $target_s$ from it, and then line 4 in figure 3.5 is finished.

(a) The case that in an unmerged cluster there is the immediate successor task of $btm_s(p)$, where $btm_s(p) = \{n_{p'}^s\}$, privided that $pivot_s = \{cls_s(p)\}$. (figure 3.7 (a)).
In immediate successor tasks of $btm_s(p)$, where $btm_s(p) = \{n_{p'}^s\}$, let the task in an unmerged cluster be $n_{u'}^s$, and let the cluster to which $n_{u'}^s$ belong be $cls_s(u)$, i.e., we have $cls_s(u) = \{n_{u'}^s\}$. Then let such a set of $cls_s(u)$ be $TGT_c(cls_s(p))$. If we define the cluster $cls_s(t)$ in which tasks satisfying the following condition be $target_s$, which is selected from $TGT_c(cls_s(p))$.

$$
\begin{aligned}
& c(e_{p',t'}^s) + blevel(n_{t'}^s) = \max_{\substack{n_{u'} \in cls_s(u), \\ cls_s(u) \in TGT_s(cls_s(p))}} \left\{ c(e_{p',u'}^s) + blevel(n_{u'}^s) \right\}, \\
& where\ n_{t'}^s \in cls_s(t), cls_s(t) \in TGT_c(cls_s(p)), btm_s(p) = \{n_{p'}^s\}, \\
& TGT_c(cls_s(p)) = \{cls_s(u) | cls_s(u) = \{n_{u'}^s\}, n_{u'}^s \in suc(n_{p'}^s)\}. \qquad (3.34)
\end{aligned}
$$

In this case, after the task merging step, $cls_{s+1}(p)$ is linear, so that every precedence relationship among any tasks in $cls_{s+1}(p)$ is maintained. This fact satisfies the condition 2. We have $-c(e_{p',u'}^s) \leq BL_{s+1}(p) - BL_s(p) \leq 0$ by the task merging step with $target_s$ being selected based on eq.(3.34). That is, $LV_{s+1}(p)$ is not increased by the task merging step, i.e., $LV_s(p) \geq LV_{s+1}(p)$.

(b) The case that $pivot_s = \{cls_s(p)\}$ is linear and every immediate successor task of $btm_s(p)$ where $btm_s(p) = \{n_{p'}^s\}$ belongs to a cluster having two or more tasks (figure 3.7 (b)).
In this case, if $target_s$ is selected by tracing immediate successor tasks of $btm_s(p)$ based on the policy (a), it is possible that $cls_{s+1}(p)$ becomes non-linear after a task merging step. Thus, the objective in this case is to minimize the increase of $BL_{s+1}(p)$ based on condition 1. From the definition in table 3.1, $BL_s(p)$ is derived by tasks in $out_s(p)$. Thus, let define the task which has immediate successor tasks in an unmerged cluster be n_k^s. Then, assume such a set of tasks n_k^s is $out'_s(p)$ and then the algorithm select the task which has the maximum $S(n_k^s, p) + blevel(n_k^s)$ value in $out'_s(p)$ (let the task be $n_{p'}^s$). Moreover, let the set of unmerged clusters to which immediate successor tasks of $n_{p'}^s$

belong as $TGT_c(cls_s(p))$. Let a cluster $cls_s(t)$ to which tasks satisfying the following condition be $target_s$.

$$
\begin{aligned}
&c(e^s_{p',t'}) + blevel(n^s_{k'}) = \max_{\substack{n^s_{u'} \in cls_s(u), \\ cls_s(u) \in TGT_c(cls_s(p))}} \left\{ c(e^s_{p',u'}) + blevel(n^s_{u'}) \right\}, \\
&where\ n^s_{t'} \in cls_s(t), cls_s(t) \in TGT_c(cls_s(p)), n^s_{t'} \in suc(n^s_{p'}), \\
&S(n^s_{p'}, p) + blevel(n^s_{p'}) = \max_{n^s_k \in out'_s(p)} \left\{ S(n^s_{k'}, p) + blevel(n^s_k) \right\}, \\
&out'_s(p) = \left\{ n^s_k | n^s_k \in out_s(p), cls(q) = \{n^s_{q'}\} for \exists n^s_{q'} \in suc(n^s_k), n^s_{q'} \in cls(q) \right\}, \\
&TGT_c(cls_s(p)) = \left\{ cls_s(u) | cls_s(u) = \{n^s_{u'}\}, n^s_{u'} \in suc(n^s_{p'}) \right\}. \qquad (3.35)
\end{aligned}
$$

If we define $\Delta BL_{s+1}(p) = BL_{s+1}(p) - BL_s(p)$, by the localization of $e^s_{p',t'}$ we have $-c(e^{s+1}_{p',t'}) \leq \Delta BL_{s+1}(p)$ in case of $BL_s(p) = S(n^s_{p'}, p) + blevel(n^s_{p'})$. On the other hand, we have $n^s_{t'} \in cls_s(p)$ after the task merging step, and then we have $\Delta BL_{s+1}(p) \leq w(n^s_{t'})$. Hence, the increase of $LV_{s+1}(p)$ after a task merging step is within from $-c(e^s_{p',t'})$ to $w(n^s_{t'})$.

(c) The case that $pivot_s = \{cls_s(p)\}$ is not linear and there are immediate successor tasks of $out_s(p)$, and they are included in unmerged clusters (figure 3.7 (c)).
Since $cls_s(p)$ is not linear, it is not linear after a task merging step. Even if an unmerged cluster, in which a task is an immediate successor task of $btm_s(p)$, is selected as $target_s$, the number of tasks without precedence relationships may be increased after a task merging step. Hence, this case does not satisfy condition 2. Thus, the objective of this procedure is to minimize the increase of $BL_{s+1}(p)$ by tracing immediate successor tasks of $out_s(p)$. Similar to (b), let the task which is an immediate successor task of un unmerged cluster be n^s_k, and then let the set of n^s_k be $out'_s(p)$. Then the algorithm selects the task having the maximum of $S(n^s_k, p) + blevel(n^s_k)$ value (let the task be $n^s_{p'}$). Moreover, in clusters to which immediate successor tasks of $n^s_{p'}$ belong, let an unmerged cluster be $cls_s(u) = \{n^s_{u'}\}$ and let the set of $cls_s(u)$ be $TGT_c(cls_s(p))$. Then the cluster $cls_s(t)$ satisfying the following condition is selected from $TGT_c(cls_s(p))$.

$$
\begin{aligned}
&c(e^s_{p',t'}) + blevel(n^s_{k'}) = \max_{\substack{n^s_{u'} \in cls_s(u), \\ cls_s(u) \in TGT_c(cls_s(p))}} \left\{ c(e^s_{p',u'}) + blevel(n^s_{u'}) \right\}, \\
&S(n^s_{p'}, p) + blevel(n^s_{p'}) = \max_{n^s_k \in out'_s(p)} \left\{ S(n^s_{k'}, p) + blevel(n^s_k) \right\}, \\
&out'_s(p) = \left\{ n^s_k | n^s_k \in out_s(p), cls(q) = \{n^s_{q'}\} for \exists n^s_{q'} \in suc(n^s_k), n^s_{q'} \in cls(q) \right\}, \\
&TGT_c(cls_s(p)) = \left\{ cls_s(u) | cls_s(u) = \{n^s_{u'}\}, n^s_{u'} \in suc(n^s_{p'}) \right\}. \qquad (3.36)
\end{aligned}
$$

As a result, similar to (b), the increase of $LV_{s+1}(p)$ is within from $-c(e^s_{p',t'})$ to $w(n^s_{u'})$, since $n^{s+1}_{t'}$ is included in $cls_{s+1}(p)$.

(d) The case every immediate successor task of $out_s(p)$ belong to a cluster having two or more tasks, irrespective of whether $pivot_s = \{cls_s(p)\}$ is linear or not (figure 3.7 (d)).
In this case, the objective of the algorithm is to minimize the increase of $LV_{s+1}(p)$, since there is no unmerged cluster even if every immediate successor task of $out_s(p)$

is traced. If $cls_s(p)$ is linear, it is preferable that $cls_{s+1}(p)$ is also linear. To do this, it is necessary to select $target_s$ from a cluster which is not $cls_s(p)$ and to which an immediate predecessor task of $in_s(p)$ belongs. From the definition in table 3.1, since $in_s(p)$ has both tasks included in $top_s(p)$ and tasks not included in $top_s(p)$, at first it is necessary to decide which immediate predecessor tasks should be traced in $in_s(p)$ and $top_s(p)$. Let assume that the task in which $in_s(p)$ does not belong to $top_s(p)$, and the cluster to which an immediate predecessor task of n_k^s belongs is $target_s$. In this case, there may be a task in $target_s$ which has no precedence relationship with $top_s(p)$. As a result, after a task merging step, $|top_{s+1}(p)|$ is increased and then $LV_{s+1}(p)$ is also increased, i.e., $LV_s(p) < LV_{s+1}(p)$. This is because every task in top_s is executed before other tasks in the same cluster, so that $BL_{s+1}(p)$ is increased, i.e., $BL_s(p) < BL_{s+1}(p)$. Thus, to maintain precedence relationships in $cls_{s+1}(p)$, it is necessary to select the cluster $target_s$ to which an immediate predecessor task of $top_s(p)$ belongs.

Next let describe about the criterion to reduce $LV_{s+1}(p)$, i.e., $LV_s(p) > LV_{s+1}(p)$, or minimize the increase of it. If $cls_{s+1}(p)$ is generated based on either (a) or (b) or (c), we have $|top_{s+1}(p)| = 1$ (as for the case the number of tasks in $top_{s+1}(p)$ is increased, we describe in (e)). As a result, if $|top_s(p)| = 1$ before a task merging step, $TL_{s+1}(p)$ is equal to TL_s value of $target_s$ which is the cluster to which an immediate predecessor task of $top_s(p)$ belong, because $top_{s+1}(p)$ is the set of tasks in top_s of $target_s$. Thus, the objective is to minimize the increase of LV_{s+1} at $cls_{s+1}(p)$. For any task $n_{p'}^s \in top_s(p)$ before a task merging step, let the immediate predecessor task of $n_{p'}^s$ be $n_{u'}^s$, and assume $n_{u'}^s \in cls_s(u)$. Assume the set of $cls_s(u)$ is defined as $TGT_c(cls_s(p))$, and then the cluster $cls_s(t)$ having the maximum LV_s value in $TGT_c(cls_s(p))$ is defined as $target_s$.

$$
\begin{aligned}
&LV_s(t) = \max_{cls_s(u) \in TGT_c(cls_s(p))} \{LV_s(u)\}, \\
&where\ n_{t'}^s \in pred(n_{p'}), n_{t'}^s \in cls_s(t), cls_s(t) \in TGT_c(cls_s(p)), \\
&TGT_c(cls_s(p)) = \{cls_s(u) | n_{u'}^s \in pred(n_{p'}^s), n_{u'}^s \in cls_s(u)\}, n_{p'}^s \in top_s(p), \\
&TL_s(p) = tlevel(n_{p'}^s).
\end{aligned} \tag{3.37}
$$

In this case, the increase of $LV_{s+1}(p)$ after a task merging step depends on not only TL_s of $target_s$ before a task merging step, but also precedence relationships among tasks dominating $BL_s(p)$.

(e) $top_s(p)$ has only a START task and every immediate successor task of $out_s(p)$ belongs to a cluster having two or more tasks, irrespective of $pivot_s = \{cls_s(p)\}$ is linear or not (figure 3.7 (e)).
Since $top_s(p)$ has only a START Task, $target_s$ can not be selected by policies of figure 3.7 (a), (b), (c), and (d), i.e., there is no candidate for $target_s$. Since the number of tasks in $target_s$ is two or more, from theorem 3.8 tasks without precedence relationships are included in $cls_{s+1}(p)$ by the task merging step between $cls_s(p)$ and $target_s$. On the other hand, since $top_s(p)$ has only a START task, we have $TL_s(p) = 0$. Moreover, from theorem 3.8 the number of tasks in $top_{s+1}(p)$ is increased by a task merging step. As a result, $TL_{s+1}(p)$ is equal to TL_s value of $target_s$, and $BL_{s+1}(p)$ depends on precedence relationships among tasks which belong to $cls_{s+1}(p)$. Thus, the objective in this case is to minimize the increase of $BL_{s+1}(p)$ after a task merging step.

As eq.(3.38) presented below, the algorithm specifies the task having the maximum $S(n^s_{k'}, p) + blevel(n^s_{k'})$ value in the set of $n^s_{k'}$ each of which belongs to $out_s(p)$. Then the algorithm selects the cluster $cls_s(t)$ to which immediate successor tasks (defined as $n^s_{t'}$) of $n^s_{p'}$ which dominate $blevel(n^s_{p'})$ belong as $target_s$.

$$
\begin{aligned}
& c(e^s_{p',t'}) + blevel(n^s_{t'}) = \max_{\substack{n^s_{u'} \in cls_s(u), \\ cls_s(u) \in TGT_c(cls_s(p))}} \left\{ c(e^s_{p',u'}) + blevel(n^s_{u'}) \right\}, \\
& where\ BL_s(p) = S(n^s_{p'}, p) + blevel(n^s_{p'}), \\
& n^s_{p'} \in out_s(p), n^s_{t'} \in cls_s(t), cls_s(t) \in TGT_c(cls_s(p)), \\
& TGT_c(cls_s(p)) = \{cls_s(u) | n^s_{t'} \in cls_s(u), n^s_{t'} \in suc(n^s_{p'})\}. \qquad (3.38)
\end{aligned}
$$

Thus, the objective is to minimize the increase of $BL_{s+1}(p)$ by decrease $c(e^s_{p',t'})$ as much as possible.

3.4.7 Task merging steps and update procedures for merging priorities

In this section we describe procedures of $clustering(pivot_s, target_s)$ at line 5 in figure 3.5. figure 3.8 shows detailed procedures of $clustering(pivot_s, target_s)$. As defined at line 0 in figure 3.8, $pivot_s$ is the cluster $cls_s(p)$, which is selected by eq.(3.33), while $target_s$ is $cls_s(t)$ which is selected by either figure 3.7 (a) or (b) or (c) or (d) or (e). $clstering(pivot_s, target_s)$ has three kinds of procedures, i.e., task merging steps, update procedures for selection priorities of $pivot_s$, and update procedures for RDY_s.

At first, let describe a task merging step. This procedure corresponds to line 1-6 in figure 3.8. If the size of cluster of $cls_{s+1}(p)$ is δ_{opt} or more by a task merging step, $cls_{s+1}(p)$ does not belong to $RDY_{s+1} and UEX_{s+1}$. On the other hand, the cluster which was $cls_s(t)$ is removed from RDY_{s+1}, UEX_{s+1} , since $cls_s(t)$ belongs to $cls_{s+1}(p)$. By these procedures, it is guarantee that a cluster whose size is δ_{opt} or more does not belong to RDY_{s+1}.

Next, let describe update procedures for selection priorities of $pivot_s$. This procedure corresponds to line 5-25 in figure 3.8. If $target_s$ is selected by figure 3.7 (d) or (e), $TL_{s+1}(p)$ can be increased. Thus, at line 5-7, the algorithm updates both $top_{s+1}(p)$ and $TL_{s+1}(p)$. Also, if $cls_{s+1}(p)$ belongs to RDY_{s+1} by subsequent update procedures(described later) of RDY_{s+1}, it is necessary to maintain $LV_{s+1}(p)$ as the latest value. Even $cls_{s+1}(p) \notin RDY_{s+1}$, a cluster to which immediate successor tasks of $out_{s+1}(p)$ belong can be added into RDY_{s+1}. Thus, at line 8 in figure 3.8, the algorithm updates tasks in $out_{s+1}(p), btm_{s+1}(p)$ at line 8 in figure 3.8 and tasks in $out_{s+1}(p)$ at line 10 in figure 3.8. There can be some clusters which belong to RDY_{s+1} in the set of clusters to which immediate predecessor tasks of $in_{s+1}(p)$ belong. Thus, it is necessary to update BL_{s+1} value for such clusters. Hence, the algorithm updates $blevel$ values of tasks in $in_{s+1}(p)$ at line 17-19 in figure 3.8, and then at line 23 the algorithm updates $blevel$ value for each task in such a cluster.

Finally, let describe update procedures for RDY_{s+1}. After a task merging step, there can be some clusters to which immediate successor tasks of $out_{s+1}(p)$ belong and which satisfy the condition expressed as eq.(3.31). To specify such a cluster, at line 13 in figure 3.8 the algorithm labels to each outgoin edge of tasks in $out_{s+1}(p)$ as "checked," only if we

INPUT: $pivot_s, target_s$
OUTPUT: RDY_{s+1}
0. Let $pivot_s$ $cls_s(p)$ which has been obtained by eq.(3.33);
0. Let $target_s$ $cls_s(t)$ which has been obtained by Fig. 3.7 (a) or (b) or (c) or (d) or (e);
0. Let $RDY_{candidate,s+1} \leftarrow \emptyset$;/* Set of clusters to which immediate successor tasks of $out_{s+1}(p)$ in case of $w(cls_{s+1}(p)) \geq \delta_{opt}$ */
0. $RDY_{s+1} \leftarrow RDY_s, UEX_{s+1} \leftarrow UEX_s$ and $s \leftarrow s+1$ for every cluster in both RDY_{s+1} and UEX_{s+1}.;/*Increment *s*, i.e., the state of clusters belonging to RDY_{s+1} or UEX_{s+1}.*/
/******** Task merging steps (procedures of $merge$ function is defined at eq.(2.1). ********/
1. $cls_{s+1}(p) \leftarrow merge(cls_s(p), cls_s(t))$ and remove $cls_{s+1}(t)$ from RDY_{s+1} and UEX_{s+1} if exists;
2. **IF** $w(cls_{s+1}(p)) \geq \delta_{opt}$ **THEN**
3. Remove $cls_{s+1}(p)$ from RDY_{s+1} and UEX_{s+1};
4. **END IF**
/******** Update procedures for the selection priority of $pivot_s$ ********/
5. **IF** $cls_s(t)$ has been obtained by Fig. 3.7 (d) or (e) **THEN**
6. Update $top_{s+1}(p)$ and $TL_{s+1}(p)$; /*If $TL_{s+1}(p)$ is varied, update $TL_{s+1}(p)$.*/
7. **END IF**
8. update $in_{s+1}(p)$, $out_{s+1}(p)$ and $btm_{s+1}(p)$; /*Update the set of tasks which require data communications with other clusters.*/
9. **FOR EACH** $n_{p'}^{s+1} \in out_{s+1}(p)$ **DO**
10. update $S(n_{p'}^{s+1}, p)$ and $tlevel(n_{p'}^{s+1})$;/*Update $tlevel$ value for each task.*/
11. **IF** $w(cls_{s+1}(p)) \geq \delta_{opt}$ **THEN**
12. **FOR EACH** $n_{k'}^{s+1} \in suc(n_{p'}^{s+1}), n_{k'}^{s+1} \in cls_{s+1}(k) \neq cls_{s+1}(p)$ **DO**
/*The procedure for specifying the cluster which can be added into RDY_{s+1} by tracing immediate successor tasks of $n_{p'}^{s+1}$.*/
13. mark "checked" on $e_{p',k'}^{s+1}$ and put $cls_{s+1}(k)$ into $RDY_{candidate,s+1}$;
14. **END FOR**
15. **END IF**
16. **END FOR**
17. **FOR EACH** $n_{p'}^{s+1} \in in_{s+1}(p)$ **DO**
18. update $blevel(n_{p'}^{s+1})$; /* Required for procedures at line 20, 21-25.*/
19. **END FOR**
20. Update $BL_{s+1}(p)$ by tracing $out_{s+1}(p)$; /*Required for the case that $cls_{s+1}(p) \in RDY_{s+1}$ after a task merging step.*/
21. **FOR EACH** $n_{p'}^{s+1} \in in_{s+1}(p)$ **DO**
22. **FOR EACH** $n_{k'}^{s+1} \in pred(n_{p'}^{s+1}), n_{k'}^{s+1} \in cls_{s+1}(k) \neq cls_{s+1}(p), cls_{s+1}(k) \in RDY_{s+1}$ **DO**
23. update $blevel(n_{k'}^{s+1})$; /*Required for updating $BL_{s+1}(k)$ value*/
24. **END FOR**
25. **END FOR**
/******** Update procedures for RDY_{s+1} ********/
26. **FOR EACH** $cls_{s+1}(k) \in RDY_{candidate,s+1}$ **DO**
27. **IF** $\forall n_{q'}^{s+1} \in pred(n_{k'}^{s+1}), \forall n_{k'}^{s+1} \in top_{s+1}(k)\ s.t.,\ e_{q',k'}^{s+1}$ is "checked" **THEN**
28. Update $LV_{s+1}(k)$ and put $cls_{s+1}(k)$ into RDY_{s+1};/*If $cls_{s+1}(k)$ satisfies the condition of RDY_{s+1}, it is added into RDY_{s+1}.*/
29. **END IF**
30. **END FOR**
31. **RETURN** RDY_{s+1};

Figure 3.8: Procedure of task clustering at Fig. 3.5 line. 5 (appears in [22]).

have $w(cls_{s+1}(p)) \geq \delta_{opt}$ after a task merging step. Then the cluster which has possibility to be added into RDY_{s+1} is added into $RDY_{candidate,s+1}$. At line 26-30 in figure 3.8, the algorithm traces every cluster in $RDY_{candidate,s+1}$ and check if each cluster satisfies eq.(3.31) or not. At line 27 in figure 3.8, for each task in $top_{s+1}(k)$, if every incoming edge is labeled as "checked", $cls_{s+1}(k)$ is added into RDY_{s+1}. This is because any immediate predecessor task of $top_{s+1}(k)$ belongs to a cluster whose size is δ_{opt} or more.

On subsequent selection of $pivot_{s+1}$ (at line 3 in figure 3.5), the algorithm compares LV_{s+1} values among tasks in RDY_{s+1}. Thus, at line 28 in figure 3.8 $LV_{s+1}(k)$ is updated in advance. By doing this, a cluster which is newly added into RDY_{s+1} becomes a candidate for selecting $pivot_{s+1}$. From the theorem described below, $pivot_s$ can always be selected during the algorithm, thereby we have $UEX_s = \emptyset$ by repeating a task merging step and then the algorithm can end.

Thorem 3.9. *Except the end of the algorithm, there are one or more clusters in RDY_s.*

Proof 7. *Assume $RDY_s = \{\emptyset\}$. For any task $cls_s(i)$ in V^s_{cls}, at least one immediate predecessor task of $top_s(i)$ belongs to a cluster whose size is δ_{opt} or more. Thus, let define a cluster to which immediate predecessor tasks of $top_s(i)$ belong as $cls_s(h)$, and assume $w(cls_s(h)) < \delta_{opt}$, $cls_s(h) \notin RDY_s$. Then we obtain two cases, i.e., the number of tasks in $cls_s(h)$ is "one" or "two or more."*

If $|cls_s(h)| = 1$, it corresponds to the case that the number of tasks in every cluster is one. Thus, we have $V^s_{cls} = V^0_{cls}$. However, this conflicts with that a cluster having a START task belongs to RDY_s at line 1 in figure 3.5.

If $|cls_s(h)| \geq 2$, it corresponds to the case that $cls_s(h)$ is generaged by at least one task merging step. However, from eq.(3.31) it is not true that $cls_s(h) \notin RDY_s$, since a cluster having two or more tasks belongs to RDY_s until its size becomes δ_{opt} or more.

Hence, the theorem is proved. ■

As described in sec.3.4.3, LV_{s+1} value for every cluster in RDY_{s+1} can be maintained as the latest value by tracing every task. However, due to the problem related to the time complexity, the algorithm updates only LV_{s+1} values of clusters which is newly added into RDY_{s+1}. From theorem 3.2, TL_{s+1} values of clusters in RDY_{s+1} is not varied, excpet newly added clusters, i.e., $TL_s = TL_{s+1}$. On the other hand, $blevel$ values of tasks which belong to such clusters can be decreased. Thus, BL_s value of a cluster to be selected as $pivot_s$ may not be latest, i.e, actual value can be smaller than BL_s, because the $blevel$ value of the task is decreased. However, from theorem 3.3 and 3.4, a task merging step does not have effect on BL_s values of clusters to which tasks in seq_s belong. Hence, if a task in seq_s belongs to a cluser selected as $pivot_s$ (i.e., the cluster having the maximum LV_s value in RDY_s), its LV_s value is latest. That is, such a policy does not have problem in terms of the performance in the point that it tries to decrease $sl_w(G^s_{cls})$.

3.4.8 Complexity analysis

In this section, we analyze the time complexity of the algorithm. The number a task is traced depends on the number it is merged into a cluster until the cluster size exceeds δ_{opt}. Let the

upper bound of the number be $\varepsilon(G^0_{cls})$, and then we have

$$\varepsilon(G^0_{cls}) = \left\lceil \frac{\delta_{opt}}{\min\limits_{n^0_k \in V_0} \left\{ w(n^0_k) \right\}} \right\rceil . \tag{3.39}$$

Time complexity at line 3-5 in figure 3.5 is multiplied by the number a cluster is traced, i.e., $\varepsilon(G^0_{cls})$. We describe time complexities of procedures in 3.5.

- Time complexity of $pivot_s$ selection (line 3 in figure 3.5)
  The procedure at line 3 in figure 3.5 corresponds to the procedure for selecting $pivot_s$ from RDY_s. if each cluster in RYD_s is sorted according to nonincreasing order of LV_s value, one selection of $pivot_s$ requires only one step. Thus, from eq.(3.39) the time complexity at line 3 in figure 3.5 is $O(\varepsilon(G^0_{cls})|V|)$.

- Time complexity of $target_s$ selection (line 4 in figure 3.5)
  The procedure at line 4 in figure 3.5 corresponds to the procedure for searching $target_s$ in the order of figure 3.7 (a), (b), (c), (d), and (e). At (a) in figure 3.7, the algorithm search for the task $n^s_{t'}$ which satisfies eq.(3.34) from $suc(n^s_{p'})$. This procedure requires to sort each cluster $cls(u) = \{n^s_{u'}\}$ in $TGT_c(cls_s(p))$ at eq.(3.34) in nonincreasing order of $c(e^s_{p',u'}) + blevel(n^s_{u'})$. Thus, this requires $|suc(n^s_{p'})| \log |suc(n^s_{p'})|$ steps, and as a whole the time complexity of (a) is $O(\varepsilon(G^0_{cls})|E| \log |E|)$.

  At figure 3.7 (b), it is necessary to specify $out^{'}_s(p)$ at eq.(3.35). This procedure requires to trace both each task and each edge among tasks in $cls_s(p)$ at once, and the total number of steps is the sum of $|cls_s(p)|$ and the number of edges among tasks in $cls_s(p)$. Selecting the task $n^s_{p'}$ having the maximum $S(n^s_{k'}, p) + blevel(n^s_{k'})$ value in the set of tasks $n^s_{k'}$ in $out^{'}_s(p)$ requires $|out^{'}_s(p)| \log |out^{'}_s(p)|$ steps by the merge sorting. After that, for each task $n^s_{u'}$ in $suc(n^s_{p'})$, selecting the task having the maximum $c(e^s_{p',u'}) + blevel(n^s_{u'})$ value requires $|suc(n^s_{p'})| \log |suc(n^s_{p'})|$ steps by the merge sorting. The dominating part in the $target_s$ selection procedure at figure 3.7 (b) is the procedure for selecting $n^s_{t'}$ at eq.(3.35) if $|V| \leq |E|$. Hence, the time complexity of (b) is $O(\varepsilon(G^0_{cls})(|E| \log |E|))$. This is as same as figure 3.7 (c).

  At figure 3.7 (d), at first $n^s_{p'}$ is selected at eq.(3.37). If $TL_s(p)$ has already been decided, $n^s_{p'}$ can be selected by only one step. Then the cluster having the maximum LV_s value is selected from the set of clusters to which each task in $pred(n^s_{p'})$ belong. This requires $|pred(n^s_{p'})| \log |pred(n^s_{p'})|$ steps by the merge sorting. Hence, as a whole the time complexity of (d) is $O(\varepsilon(G^0_{cls})(|E| \log |E|))$.

  At figure 3.7 (e), at first $n^s_{p'}$ is selected at eq.(3.38). If $BL_s(p)$ has already been decided, $n^s_{p'}$ is selected by only one step. Then the task having the maximum $c(e^s_{p',u'}) + blevel(n^s_{u'})$ value in tasks $n^s_{u'}$ which belong to $suc(n^s_{p'})$. This requires $|suc(n^s_p)| \log |suc(n^s_p)|$ steps by the merge sorting. Thus, the time complexity of (d) is $O(\varepsilon(G^0_{cls})(|E| \log |E|))$. The time complexity at line 4 in figure 3.5 is $O(\varepsilon(G^0_{cls})(|E| \log |E|))$.

- Time complexity of a task merging step and update procedures for merging priorities (line 5 in figure 3.5)

The procedure at line 5 in figure 3.5 corresponds to the procedure in figure 3.8. At line 2-4 in figure 3.8, it is necessary to specify $cls_s(p)$ from RDY_s, UEX_s, i.e., the time complexity of this is $O(\varepsilon(G^0_{cls}) \log |V|)$.

At line 5-7 in figure 3.8, the set of tasks which belong to $top_{s+1}(p)$. This requires toi trace each task and each edge among tasks in $top_s(p) \cup top_s(t)$ at once, as a whole the time complexity is $O(\varepsilon(G^0_{cls})(|V| + |E|))$.

Then $tlevel$ values of tasks in $top_{s+1}(p)$ is updated by tracing their incoming edges. Thus, as a whole the time complexity is $O(\varepsilon(G^0_{cls})(|V| + |E|))$. After that, from $tlevel$ vlaues of tasks in $top_{s+1}(p)$, it is necessary to decide the maximum, i.e., $TL_{s+1}(p)$. This requires $|top_{s+1}(p)| \log |top_{s+1}(p)|$ steps by the merge sorting, and Hence as a whole the time complexity is $O(\varepsilon(G^0_{cls})|V| \log |V|)$.

The time complexity at line 5-7 in figure 3.8 is $O(\varepsilon(G^0_{cls})|V| \log |V|)$. At line 8 in figure 3.8, the procedure traces both incoming edges and outgoing edges among tasks in $cls_{s+1}(p)$ after a task merging step, whose time complexity is $O(\varepsilon(G^0_{cls})|E|)$. It is necessary to specify the set of tasks without precedence relationships with $n^{s+1}_{p'}$ for updating $tlevel$ value of each task $n^{s+1}_{p'}$ in $out_{s+1}(p)$ at line 10 in figure 3.8. This requires to trace tasks and edges among them in $cls_{s+1}(p)$ for each task in $out_{s+1}(p)$, so that as a whole the time complexity is $O(\varepsilon(G^0_{cls})|V|(|V| + |E|))$. Then it is necessary to add size of each task which precedes $n^{s+1}_{p'}$ in $cls_{s+1}(p)$ in order to update $tlevel$ value for each task in $out_{s+1}(p)$, which requires $|out_{s+1}(p)|$ steps. The time complexity of the procedure at line 10 in figure 3.8 is $O(\varepsilon(G^0_{cls})|V|(|V| + |E|))$. At line 12-14 in figure 3.8, it is necessary to trace outgoing edges of each task in $out_s(p)$, whose time complexity is $O(\varepsilon(G^0_{cls})|E|)$.

At line 18 in figure 3.8, each task in edges among tasks in $cls_s(p)$ is traced at once, and then such a task 's $bevel$ value is updated, whose time complexity is $O(\varepsilon(G^0_{cls})(|V| + |E|))$.

At line 21-25 in figure 3.8, for each task $n^{s+1}_{p'}$ such that $n^{s+1}_{p'} \in in_{s+1}(p)$, $blevel$ value for each task $n^{s+1}_{k'}$ in $pred(n^{s+1}_{p'})$ is updated. Thus, it is necessary to trace tasks and incoming edges among them in $in_{s+1}(p)$, and tasks and edges among them in $pred(n^{s+1}_{p'})$, whose time complexity is $O(\varepsilon(G^0_{cls})(|V| + |E|))$.

At line 26-30 in figure 3.8, for each cluster $cls_{s+1}(k)$ in $RDY_{candidate,s+1}$, each task in $n^{s+1}_{k'} \in top_{s+1}(k)$ is checked whether it is added into RDY_{s+1} or not, which requires $|pred(n^{s+1}_{k'})|$ steps. If every task in RDY_{s+1} is sorted by the nonincreasing order of LV_{s+1}, a cluster is added into RDY_{s+1} by $\log |RDY_{s+1}|$ steps. Thus, as a whole, the time complexity is $O(\varepsilon(G^0_{cls})|E| \log |V|)$. Hence, the time complexity at line 5 in figure 3.5 is dominated by the procedure at line 10 in figure 3.8, whose time complexity is $O(\varepsilon(G^0_{cls})|V|(|V| + |E|))$.

From the analysis described above, the time complexity of the algorithm is dominated by the procedure at line 5 in figure 3.5, whose time complexity is $O(\varepsilon(G^0_{cls})|V|(|V| + |E|))$.

3.5 Experimental comparison

In this section, we present the experimental comparison results by simulations.

3.5.1 Comparison points

Objectives of this comparison is to confirm that decreasing $sl_w(G^R_{cls})$ based on δ_{opt} can lead to the decrease of the schedule length $sl(G^R_{cls})$, and that how effectively each PE can be utilized by the proposal. Here, R satisfies eq.(3.1). Thus, we compared the following points.

1. Comparison of the combination of tasks for each cluster.
2. Comparison of both $sl_w(G^R_{cls})$ and $sl(G^R_{cls})$ when CCR [16, 41] is varied according to a DAG, where every approach has the same number of generated clusters.
3. Comparison of $sl(G^R_{cls})$ with CCR being varied (the number of generated clusters is varied).
4. Comparison of the schedule length by different scheduling policies.
5. Evaluation of the optimality of δ_{opt}.
6. Comparison of algorithm running time among several task clustering algorithms.
7. Comparison of effective use of processors (defined in sec.3.5.10) using a realistic application DAG.

As one indication representing characteristics of a DAG, CCR(Communication to Computation Ratio) [16, 41] is well known. CCR is defined as the ratio of data size among tasks to the task size, or as the ratio of the average data size to the average task size. Various kinds of DAGs can be defined by changing CCR from 0.1 to 10.0 [16, 41]. In this experiment, a DAG is generated by changing CCR.

One objective of the proposed task clustering is to minimize the schedule length with small number of PEs by doing task merging steps until each cluster size is δ_{opt} or more. At (1), the comparison is about the combination of tasks by which $sl_w(G^R_{cls})$ is minimized, which is one objective of the proposal. From this comparison, we prove the reason why $sl_w(G^R_{cls})$ is reduced (or increased).

At (2), $sl_w(G^R_{cls})$ and $sl(G^R_{cls})$ are compared among several approaches by the number of PEs decided by the proposed task clustering. However, whether the number of PEs decided at (2) is optimal for minimizing the schedule length or not is unknown. Thus, at (3) it is confirmed that how the number of PEs decided by the proposal and $sl(G^R_{cls})$ is optimal. That is, we confirm that how the number of PEs obtained by the proposal is small with compared to that obtained by conventional approaches with the same $sl(G^R_{cls})$ value. Also, the schedule length depends on the execution order for each task, i.e., the scheduling policy. Hence, at (4) we compared the schedule length by several scheduling policies after the proposed task clustering. By using the result at (4), we prove the relationship between the proposed task clustering and the scheduling policy. At (5), how the schedule length can be reduced by the lower bound δ_{opt} with compared to other lower bounds is evaluated. At (6), we compared each running time by several task clustering algorithms. At (7), we compared the degree of utilization for each PE in cases of Gaussian Elimination (GE) DAG [34] and FFT DAG [36] in order to evaluation the practicability of the proposal.

Table 3.3: Configuration policies for each parameter in a random DAG (appears in [22]).

Parameter	Policy for Assigning Value
$w(n_k^0), n_k^0 \in V_0$	Max/Min Ratio = 100
$c(e_{k,l}^0), e_{k,l}^0 \in E_0$	Max/Min Ratio = 100
Parallelism Factor (PF) [38,39]	$\frac{\sqrt{\lvert V_0 \rvert}}{\alpha}$, where α is selected randomly from 0.5, 1.0 and 2.0
Out Degree of each Task	Randomly selected from 1 to 5 for each Task.
CCR [16,41]	Each random DAG's CCR is within $[0.1, 10]$.

3.5.2 Simulation environment

As for sec.3.5.1 (1) to (6), we generated a random DAG as an input data, and at (7) we generated a GE DAG [34] and FFT DAG [36].

At first, we describe how to generate a random DAG. Table 3.3 shows the policy for generating a random DAG. When a random DAG is generated, we set the ratio of maximum to minimum of each task size and each data size as 100. At (1) we set those parameters by uniform distribution, at (2) we set them by uniform distribution and normal distribution, at (3) to (6) we set them by uniform distribution.

The degree of parallelism of a DAG is defined as α by Parallelism Factor (PF), while the depth, i.e., the maximum number of tasks on one path, is defined as $\frac{\sqrt{|V_0|}}{\alpha}$ [38, 39]. α is set from the set of 0.5, 1.0, 2.0 [39], by which the depth of a DAT according to uniform distribution is decided. Also, as the literature [39], we generated a DAG by deciding the number of outgoing edges for each task from 1 to 5 according to uniform distribution.

In a task scheduling is performed, one critical issue is to how to minimize the communication latencies among tasks, especially when each task in an application which requires large number of communication is scheduled. Thus, each data size does not depend on the number of edges on generating a random DAG. In this experiment, CCR is defined as the ratio of the average data size to the average task size, and CCR is within 0.1 to 10.0.

The simulation environment was developed by J2SE1.6_03. The simulation runtime environment is, JRE1.6.0_03, OS is Windows XP SP3, CPU is Intel Core 2 Duo 2.66GHz, and memory size is 2.0GB.

3.5.3 Comparison targets

Comparison targets in sec.3.5.1 is decided based on two criteria, i.e., 1. cluster merging is performed after a task clustering, 2. only cluster merging (task merging steps) is performed. As the first criterion, the one approach is that Load Balancing (LB) [26] is performed after the task clustering by CASS-II [29] (CASS-II+LB), and the other is that Cluster Merging(CM) is performed after the task clustering by DSC [18] (DSC+CM) [25]. On the other hand, as the second criterion there are two approaches, i.e., the proposal and LB.

Throughout this section, assume that every cluster size becomes $\delta = \delta_{opt}$ or more after R task merging steps have been performed. On the other hand, at experiments by sec.3.5.1 (2), (3), (4), (6), and (7), conventional approaches perform task merging steps until the specified number of clusters (PEs) is reached.

From eq.(2.1), we can see that the number of clusters is decremented by one task merging step. Thus, if the number of clusters generated by R task merging steps of the proposal is equal to that of conventional approaches, the number of task merging steps of those conventional approaches is also R. Of course every cluster size after R task merging steps by conventional approaches is not always δ_{opt} or more. Hence, we commonly denote the DAG after R task merging steps for every approaches as G^R_{cls}.

At sec.3.5.1 (2), (3), (5), and (7), $sl(G^R_{cls})$ of each approach is derived by RCP(Ready Critical Path) [28] scheduling algorithm.

3.5.4 Comparison of the combination of tasks for each cluster

Comparison of precedence relationships among tasks

The objective of the algorithm described in sec.3.4 is to minimize $sl_w(G^R_{cls})$ while every cluster size is δ_{opt} or more. At sec.3.4.1 (ii), we described that it is important to maintain precedence relationships among tasks as a task merging policy. Thus, for each generated cluster, we evaluate the degree of precedence relationships among tasks for each cluster. From definition in table 3.1, $sl_w(G^s_{cls})$ is derived from $LV_s(i)$ of a cluster $cls_s(i)$. Moreover, for each task n^s_k in $cls_s(i)$, if independent tasks with n^s_k are included in $cls_{s+1}(i)$ by a task merging step, $S(n^{s+1}_k, i)$ is increased, i.e., $S(n^s_k, i) < S(n^{s+1}_k, i)$.

$tlevel(n^{s+1}_k)$ is also increased, i.e., $tlevel(n^s_k) < tlevel(n^{s+1}_k)$. Especially, if $tlevel$ values of tasks in $out_{s+1}(i)$ is increased, TL_{s+1} values of clusters can also be increased, i.e., $TL_s < TL_{s+1}$. Thus, we compared that the maximum number of tasks which are executed before every task in $out_R(i)$ for each cluster $cls_R(i)$. To do this, a random DAG is generated according to the policy described in sec.3.5.2, $N_{S,ave}$ defined as follow is used for the comparison with conventional approaches.

$$N_{S,ave} = \frac{1}{w_{average}} \sum_{cls_s(i) \in V^R_{cls}} \sum_{n^R_k \in out_s(i)} \frac{S(n^R_k, i)}{|out_s(i)|}. \tag{3.40}$$

$w_{average}$ is the average of task size, and $N_{S,ave}$ is the average number of tasks included in $S(n^R_k, i)$ for each task n^R_k in each cluster $cls_R(i)$. If this value becomes larger, more independent tasks are included in the same cluster, which leads to the increase of $tlevel$ of tasks in $out_R(i)$, i.e., the increase of $BL_R(i)$. Figure 3.9 shows the comparison results of $N_{S,ave}$. As can be seen from figure 3.9, $N_{S,ave}$ value by the proposal is the lowest. Thus, it can be concluded that the proposal can merge tasks with precedence relationships in the same cluster as far as possible.

Comparison of the number of tasks which can be executed at the earliest start time for each cluster

For each cluster $cls_s(i)$, if multiple tasks belong to $top_s(i)$, they have no precedence relationships each other. Thus, if we have $|top_{s+1}(i)| > |top_s(i)|$ by a task merging step, $tlevel$ values of tasks except $top_{s+1}(i)$ can be increased. It is conceivable that this fact has effect on the increase of $sl_w(G^R_{cls})$. We compared the number of tasks in top_S for each cluster obtained

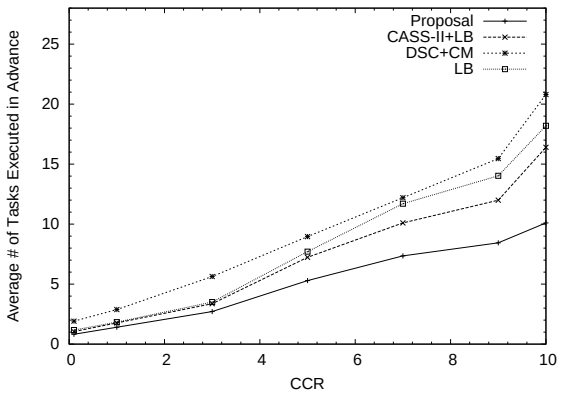


Figure 3.9: Comparison of $N_{S,ave}$ (appears in [22]).

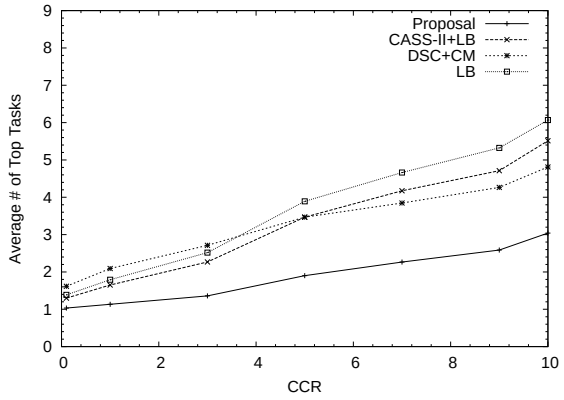


Figure 3.10: Comparison of $|top_R|$ for each cluster
(appears in [22]).

by both the proposal and conventional approaches. Figure 3.10 shows the comparison results of the number of tasks in top_S for each cluster. In the figure, y-axis corresponds to the average number of tasks in top_S for each cluster. As can be seen from the figure, though the number of tasks top_S is around 1 with low CCR value for every approach, it is increased according to larger CCRs. In the case of the proposal, the number of tasks in top_S is increased at figure 3.7 (e). In the proposal, it checks whether $target_s$ can be selected or not by one of conditions, i.e., figure 3.7 (a) or (b) or (c) or (d). If not, the algorithm selects $target_s$ by (e).

By using such a policy, the number of tasks in top_{s+1} can not be increased, so that $tlevel$ values of tasks except $top_{s+1}(i)$ can be minimized. As a result, $tlevel$ values for each task in V_{cls}^R is minimized as much as possible, and then it is conceivable that $N_{S,ave}$ value in figure 3.9 becomes lower than that of conventional approaches.

3.5.5 Comparison of WSL and the schedule length with changing CCR

In this experiment, we compared $sl_w(G_{cls}^R)$ and $sl(G_{cls}^R)$ in a DAG with CCR being varied. In the proposal, the number of clusters, i.e., the number of PEs can be decided after task merging steps have been performed in order to achieve that every cluster size is δ_{opt} or more. Under those conditions, 100 random DAGs are generated and then we compared both averaged $sl_w(G_{cls}^R)$ and averaged $sl(G_{cls}^R)$. CCR is decided by each task size and each data size. We performed the evaluation with assigning two patterns of random values to each task size and data size, i.e., uniform distribution and normal distribution.

Table 3.4 shows the comparison results in the case that both each task size and each data size are set according to uniform distribution.

The number of tasks $|V|$ is set by 500 or 1000, and we changed CCR. The column "$|V_{cls}^R|$" means the number of clusters obtained by taks merging steps with every cluster size being δ_{opt} or more. The column "$sl_w(G_{cls}^S)$ Ratio to A" means the ratio of $sl_w(G_{cls}^S)$ values of conventional approaches to that of the proposal. From table 3.4, $sl_w(G_{cls}^S)$ obtained by the proposal is lower than that of other approaches. Even if CCR is very low such as No.1 and

Table 3.4: Comparison of $sl_w(G^R_{cls})$ and $sl(G^R_{cls})$ with varying CCR (Each task size and data size is assigned according to random value in uniform distribution)
(appears in [22]).

No.	$\|V\|$	CCR	$\|V^R_{cls}\|$	$sl_w(G^S_{cls})$ Ratio to A				$sl(G^S_{cls})$ Ratio to A			
				A	B	C	D	A	B	C	D
1	500	0.1	147	1.000	1.235	1.517	1.422	1.000	1.008	1.112	1.032
2	500	1.0	45	1.000	1.315	1.506	1.359	1.000	1.192	1.230	1.274
3	500	3.0	26	1.000	1.219	1.301	1.416	1.000	1.162	1.065	1.555
4	500	5.0	22	1.000	1.237	1.432	1.579	1.000	1.281	1.334	1.797
5	500	8.0	17	1.000	1.268	1.339	1.435	1.000	1.186	1.288	1.813
6	500	10.0	16	1.000	1.348	1.340	1.401	1.000	1.280	1.438	1.949
7	1000	0.1	271	1.000	1.316	1.439	1.506	1.000	1.021	1.114	1.042
8	1000	1.0	82	1.000	1.382	1.484	1.342	1.000	1.176	1.185	1.340
9	1000	3.0	48	1.000	1.222	1.442	1.429	1.000	1.219	1.326	1.614
10	1000	5.0	38	1.000	1.374	1.391	1.443	1.000	1.327	1.262	1.688
11	1000	8.0	31	1.000	1.237	1.288	1.378	1.000	1.248	1.350	1.796
12	1000	10.0	28	1.000	1.272	1.324	1.377	1.000	1.231	1.477	1.916

7 (CCR=0.1), $sl(G^R_{cls})$ by the proposal is low, while the difference with other approaches is lower than other cases. In the case of CCR=0.1, the number of generated clusters is higher than other cases. Thus, the number of tasks for each cluster is lower than other cases, this leads to the fact that the degree of parallelism is also lower. Hence, in this case parallelism is retained and data transfer latencies have a little effect on the schedule length for every approach. We can see that a good schedule length can be obtained for every approach than other cases, by which it can be concluded that each schedule length is asymptotic. As a whole, we found that the schedule length by the proposal is lower than other approaches with the decided number of clusters by the proposal.

Next, we present the comparison result in the case that each task size and each data size are assigned by normal distribution. It is conceivable that the distribution of those values has a deviation.

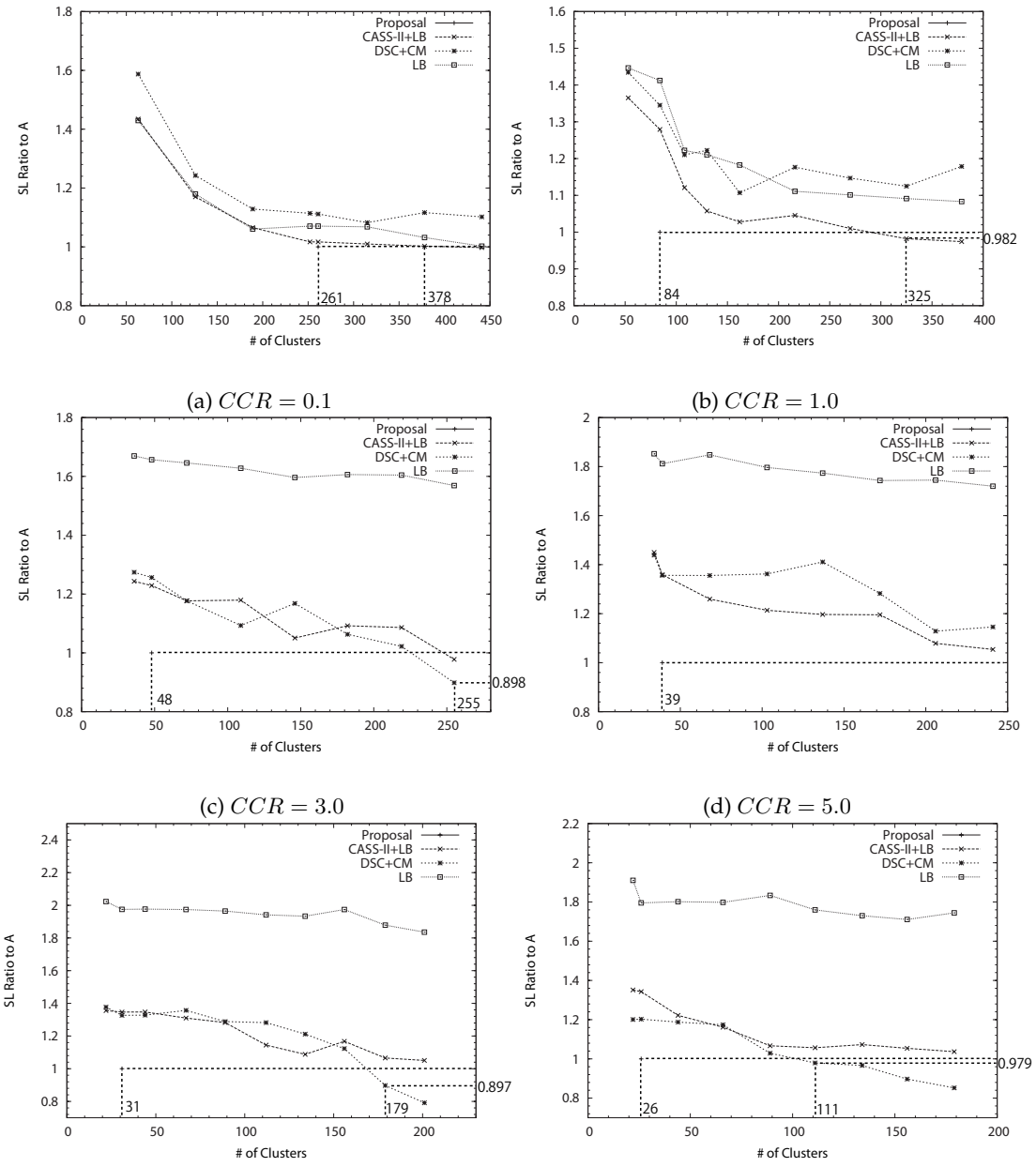
Thus, we firstly decided the maximum value and the minimum value of task size and data size. These values follow to table 3.3 and the ratio of the maximum value to the minimum value is 100 in advance. Then we decided the standard deviation $\sigma_{task}, \sigma_{data}$ and the average μ_{task}, μ_{data} as follows.

$$\begin{aligned}
\mu_{task} &= \min_{n^0_k \in V_0} \left\{ w(n^0_k) \right\} + \left(\max_{n^0_k \in V_0} \left\{ w(n^0_k) \right\} - \min_{n^0_k \in V_0} \left\{ w(n^0_k) \right\} \right) \alpha, \\
\mu_{data} &= \min_{e^0_{k,l} \in E_0} \left\{ c(e^0_{k,l}) \right\} + \left(\max_{e^0_{k,l} \in E_0} \left\{ c(e^0_{k,l}) \right\} - \min_{e^0_{k,l} \in E_0} \left\{ c(e^0_{k,l}) \right\} \right) \beta, \\
\sigma_{task} &= \frac{1}{3} \max \left\{ \mu_{task} - \min_{n^0_k \in V_0} \left\{ w(n^0_k) \right\}, \max_{n^0_k \in V_0} \left\{ w(n^0_k) \right\} - \mu_{task} \right\}, \\
\sigma_{data} &= \frac{1}{3} \max \left\{ \mu_{data} - \min_{e^0_{k,l} \in E_0} \left\{ c(e^0_{k,l}) \right\}, \max_{e^0_{k,l} \in E_0} \left\{ c(e^0_{k,l}) \right\} - \mu_{data} \right\}. \qquad (3.41)
\end{aligned}$$

As a next step, we derived $sl_w(G^R_{cls})$ and $sl(G^R_{cls})$ with changing both α and β as 0.1,

Table 3.5: Comparison of $sl_w(G^R_{cls})$ and $sl(G^R_{cls})$ with varying CCR (Each task size and data size is assigned according to random value in normal distribution, and A. Proposal, B. CASS-II+LB, C. DSC+CM, D. LB)
(appears in [22]).

No.	$\|V\|$	α	β	CCR	$\|V^R_{cls}\|$	$sl_w(G^S_{cls})$ Ratio to A				$sl(G^S_{cls})$ Ratio to A			
						A	B	C	D	A	B	C	D
1	1000	0.1	0.1	0.1	205	1.000	1.510	1.450	1.366	1.000	1.049	1.104	1.056
2	1000	0.1	0.1	3.0	39	1.000	1.197	1.351	1.348	1.000	1.163	1.183	1.571
3	1000	0.1	0.1	10.0	22	1.000	1.161	1.207	1.269	1.000	1.335	1.329	1.898
4	1000	0.1	0.9	0.1	276	1.000	1.285	1.451	1.333	1.000	1.020	1.090	1.024
5	1000	0.1	0.9	3.0	59	1.000	1.277	1.490	1.389	1.000	1.214	1.313	1.522
6	1000	0.1	0.9	10.0	34	1.000	1.415	1.412	1.475	1.000	1.398	1.519	1.966
7	1000	0.5	0.5	0.1	300	1.000	1.270	1.365	1.492	1.000	1.000	1.073	1.027
8	1000	0.5	0.5	3.0	56	1.000	1.256	1.567	1.599	1.000	1.187	1.240	1.632
9	1000	0.5	0.5	10.0	30	1.000	1.412	1.311	1.374	1.000	1.256	1.393	1.890
10	1000	0.9	0.1	0.1	251	1.000	1.484	1.375	1.383	1.000	1.019	1.086	1.048
11	1000	0.9	0.1	3.0	47	1.000	1.197	1.310	1.380	1.000	1.128	1.208	1.540
12	1000	0.9	0.1	10.0	25	1.000	1.294	1.333	1.440	1.000	1.299	1.512	2.100
13	1000	0.9	0.9	0.1	357	1.000	1.236	1.448	1.458	1.000	1.002	1.048	1.022
14	1000	0.9	0.9	3.0	61	1.000	1.278	1.494	1.561	1.000	1.213	1.212	1.727
15	1000	0.9	0.9	10.0	36	1.000	1.287	1.414	1.459	1.000	1.213	1.585	2.026



(a) $CCR = 0.1$ (b) $CCR = 1.0$

(c) $CCR = 3.0$ (d) $CCR = 5.0$

(e) $CCR = 8.0$ (f) $CCR = 10.0$

Figure 3.11: Comparison of the schedule length with changing the number of PEs ($|V| = 1000$)

(appears in [22]).

0.5, and 0.9. Table 3.5 shows the comparison results. In the figure, $sl_w(G^R_{cls})$ value by the proposal is the lowest as a whole. On the other hand, as for $sl(G^R_{cls})$, the difference in the case that CCR is 1 or more is larger than that in the case that CCR is 0.1 (i.e.,No. 1, 4, 7, 10 and 13). Similar to the case that CCR is 0.1 in table 3.4, this is because that the task parallelism is retained and that the communication latency is little.

From results described above, even if a DAG has the task size and the data size with deviation, $sl(G^R_{cls})$ is low if CCR is within 0.1 to 10.0.

3.5.6 Comparison of the required PEs

In this experiment, we compared that the number of PEs obtained by the proposal and that obtained by conventional approaches with the same schedule length. Thus, firstly the number of required PEs and $sl(G^R_{cls})$ by the proposal is derived. After that, by using conventional approaches, i.e., B. CASS-II+LB, C. DSC, D. LB, the required PEs is varied in order that the experiment has two cases, i.e., the number of clusters is lower or larger in those conventional approaches than that in the proposal. Then each cluster is merged until the number of clusters reaches the required number of clusters at B, C, and D. We generated 100 random DAGS by the policy in sec.3.5.1, and then for the number of required PEs and $sl(G^R_{cls})$, those averaged values are compared. Here, let define the degree of effective use of processors as "the schedule length / the number of PEs." We compared the degree of effective use of processors among the proposal and conventional approaches.

Figure 3.11 shows the comparison result in terms of the required number of PEs and $sl(G^R_{cls})$. In the figure, DAGs are used with CCR being larger from (a) to (f). In (a) - (f), the x-axis corresponds to the number of clusters, while the y-axis corresponds to the ratio of $sl(G^S_{cls})$ of a conventional approach to that of the posposal. Also, in (a) - (f), the number of clusters obtained by the proposal is 261, 84, 48, 39, 31, 26, respectively. At (a), $sl(G^R_{cls})$ obtained by the proposal is about the same as that obtained by B (CASS+LB) with the number of PEs being 378. Hence, the degree of effective use of processors by the proposal is better than B by 378/261≒1.45 times. On the other hand, at (b) when B (CASS-II+LB) is performed with the number of PEs being 325, it becomes 0.982 times of $sl(G^R_{cls})$ by the proposal. Thus, the ratio of the degree of effective use of processorsis is 325×0.982/84≒3.80. At (c), when C is performed with the number of PEs being 255, $sl(G^R_{cls})$ is 0.898 times of that by the proposal. Thus, the ratio of the degree of effective use of processors is 255×0.898/48=4.77. We do not find that the case that $sl(G^R_{cls})$ by (d) is lower than that of the proposal. At both (e) and (f), $sl(G^R_{cls})$ by them is lower than that of the proposal when the number of PEs is 179, 111, respectively. The ratio of the degree of effective use of processors at (e) is about 5.18, and that at (f) is about 4.18.

From those results, it is found that the schedule length becomes lower than other approaches with small number of PEs.

3.5.7 Comparison of the schedule length by different scheduling policies

As described in sec.3.5.3, comparisons in sec.3.5.1 (2), (3), (5), and (7) derives the schedule length by RCP scheduling algorithm. Since the schedule length is varied according to the execution order for each task, we evaluated the effect of both the proposal and conventional

Table 3.6: Comparison of schedule length with two scheduling policies (A. Proposal, B. CASS-II+LB, C. DSC+CM, D. LB)
(appears in [22])

No.	CCR	Scheduling	$\|V_{cls}^R\|$	$sl(G_{cls}^S)$ Ratio to A			
				A	B	C	D
1	0.1	Down-Rank [18]	267	1.000	1.029	1.121	1.033
2	0.5		115	1.000	1.055	1.146	1.157
3	1		83	1.000	1.122	1.145	1.281
4	3		50	1.000	1.202	1.182	1.561
5	5		40	1.000	1.184	1.255	1.571
6	7		33	1.000	1.166	1.202	1.678
7	10		28	1.000	1.190	1.436	1.872
8	0.1	Up-Rank [27,30,39]	277	1.000	1.059	1.112	1.068
9	0.5		119	1.000	1.152	1.192	1.205
10	1		85	1.000	1.205	1.239	1.388
11	3		50	1.000	1.295	1.527	1.678
12	5		39	1.000	1.391	1.555	1.870
13	7		33	1.000	1.429	1.422	1.926
14	10		28	1.000	1.411	1.391	1.932

approaches on the schedule length. Some conventional task clustering heuristics decide the task merging priority [18,27,30]. In those conventional task clustering heuristics, the schedule length is reculculated for every task merging step. We compared several scheduling policies, which assign "Down-Rank" (the maximum path length from a START task to the target task) and "Up-Rank" (the maximum path length from the target task to a END task)[1].

Then, each task is scheduled according to nonincreasing order of Down-Rank or Up-Rank.

We generated 100 random DAGs according to table 3.3, and then the average schedule length is compared. Table 3.6 shows the comparison result. In this table 3.6, the comparison result of the schedule length when each task is scheduled according to two patterns, i.e., the nonincreasing order of Down-Rank and nonincreasing order of Up-Rank.

In every CCR, we can see that the schedule length of the proposal has the best. However, if CCR=0.1, every schedule length is asymptotic. This is because that the task parallelism is retained and the effect on communication latencies is small, similar to the case that CCR=0.1 in table 3.4.

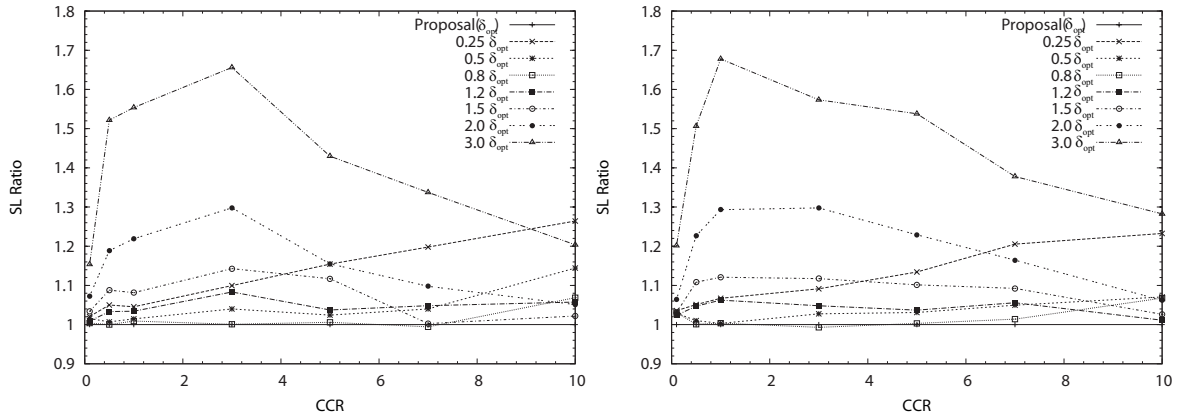
From results described above, it is found that better schedule length other than RCP can be obtained.

3.5.8 Optimality of δ_{opt}

δ_{opt} is obtained by eq.(3.22) befored the task clustering, and that is assumed to be a near-optimal value. Thus, with compared to other lower bounds for a cluster size, how δ_{opt} is better for the schedule length is evaluated.

Lower bounds for each cluster size to be compared were decided as $0.25\delta_{opt}$, $0.5\delta_{opt}$, $0.8\delta_{opt}$, $1.2\delta_{opt}$, $2\delta_{opt}$, $3\delta_{opt}$. The comparison result is shown in figure 3.12. In this figure,

[1]Some scheduling policies define other names except Down-Rank/Up-Rank. In this dissertation, we name them as Down-Rank/Up-Rank for convenience.



(a) $|V| = 500$ (b) $|V| = 1000$

Figure 3.12: Optimality about δ_{opt}
(appears in [22]).

(a) is the case of $|V| = 500$, while (b) is the case of $|V| = 1000$. The x-axis corresponds to CCR, and the y-axis corresponds to the ratio of the schedule length between the schedule length of other lower bounds to that of the proposed lower bound (i.e., δ_{opt}). If CCR=7.0 at (a), the schedule length in the case of $0.8\delta_{opt}$ is smaller than that in the case of δ_{opt}, while in other cases each schedule length is about the same, or the schedule length in the case of δ_{opt} becomes slightly smaller. The same results are obtained in the case of CCR=3.0 at (b).

Table 3.7: Breakout of ♯ of non-linear clusters after the task Clustering
(appears in [22]).

No.	$\|V\|$	CCR	♯ of non-lienar clusters on $sl_w(G^R_{cls})$/♯ of all clusters on $sl_w(G^R_{cls})$							
			$0.25\delta_{opt}$	$0.5\delta_{opt}$	$0.8\delta_{opt}$	δ_{opt}	$1.2\delta_{opt}$	$1.5\delta_{opt}$	$2.0\delta_{opt}$	$3.0\delta_{opt}$
1	500	0.1	5/30	4/23	3/19	3/16	3/14	3/11	2/7	2/5
2	500	0.5	3/26	3/17	2/13	2/11	1/9	2/7	2/5	1/3
3	500	1.0	3/20	3/14	2/11	2/8	2/6	1/5	1/3	1/2
4	500	3.0	3/15	3/12	1/8	1/6	2/5	2/4	1/3	1/1
5	500	5.0	2/13	2/10	2/7	2/5	1/4	1/3	1/2	1/1
6	500	7.0	2/11	2/8	2/6	2/4	2/4	1/2	1/2	1/1
7	500	10.0	3/10	2/7	1/5	1/4	1/3	1/2	1/1	1/1
8	1000	0.1	2/48	5/36	7/29	5/26	5/19	4/15	3/12	3/10
9	1000	0.5	5/36	6/31	4/25	3/22	2/14	2/12	2/9	2/8
10	1000	1.0	7/31	4/27	3/21	2/18	3/16	1/9	2/7	2/5
11	1000	3.0	5/26	3/22	2/18	2/15	2/13	3/8	2/5	1/3
12	1000	5.0	3/22	2/18	1/14	1/12	2/10	2/6	1/5	2/3
13	1000	7.0	3/18	3/15	2/11	2/10	2/8	2/5	1/4	1/2
14	1000	10.0	2/14	2/12	1/10	1/8	1/6	2/4	1/3	1/1

Table 3.7 shows the number of non-linear clusters in the set of clusters of $sl_w(G^R_{cls})$. Eq.(3.22) is the value of δ when $y = 1$ in eq.(3.18). The case that the lower bound is smaller than δ_{opt} corresponds to the case of $y > 1$, otherwise it corresponds to the case of $y < 1$. For

Table 3.8: Comparison of running time of each algorithm(Each task and data size is assigned according to random value in uniform distribution)
(appears in [22]).

No.	$\|V\|$	CCR	$\varepsilon(G^0_{cls})$	$\|V^R_{cls}\|$	Algorithm Running Time(ms)			
					A	B	C	D
1	500	0.1	122	147	10	17	29	29
2	500	1.0	420	45	19	33	31	38
3	500	3.0	723	26	24	29	33	38
4	500	5.0	924	22	27	29	38	36
5	500	8.0	1200	17	29	33	38	43
6	500	10.0	1287	16	29	34	38	44
7	1000	0.1	147	271	27	64	97	112
8	1000	1.0	460	82	43	88	102	140
9	1000	3.0	795	48	53	83	109	151
10	1000	5.0	1000	38	57	83	112	143
11	1000	8.0	1310	31	65	84	110	151
12	1000	10.0	1401	28	69	85	112	156

example, $0.25\delta_{opt}$ and $0.5\delta_{opt}$ can be interpreted as $y = 16$ and $y = 4$ in eq.(3.18), respectively. In table 3.7, the number of non-linear clusters is larger than 1 in the case of the lower bound is δ_{opt}. Thus, it can be said that the number of non-linear clusters is not always equal to the value of y.

At first, let describe the case that the lower bound of the cluster size is larger than δ_{opt}, i.e., $1.2\delta_{opt}$, $1.5\delta_{opt}$, $2.0\delta_{opt}$, and $3.0\delta_{opt}$. From the result in 3.12, this case has the larger schedule length than that in the case of δ_{opt} for every CCR value. The larger CCR becomes, the larger the number of independent tasks in cases of other lower bounds becomes larger than that in the case of δ_{opt}. This fact becomes one of reasons that the schedule length becomes larger with CCR. When CCR=0.1, 0.5, 1.0, 3.0 at figure 3.12 (a) and CCR=0.1, 0.5, 1 at figure 3.12 (b), the ratio of the schedule length is increasing, while in other cases the schedule length is decreasing. From those results, it can be seen that the increase of independent tasks in non-linear clusters has an impact effect on the schedule length. The larger CCR becomes, the larger the number of independent tasks is, while each communication localization has a great effect on the schedule length because of the larger lower bound for each task. As a result, it can be concluded that the ratio of the schedule length is decreased with CCR being increased.

On the other hand, from table 3.7, both the number of clusters in whith at least one task belongs to seq_R and the number of non-linear clusters are lower than the number of non-linear clusters in the case of δ_{opt}. Moreover, from figure 3.12, even if CCR is low, e.g., CCR=0.1 or 0.5, the schedule length is larger than that in the case of δ_{opt}. Even if CCR is low, the degree of task parallelism is lower than that in the case of δ_{opt} due to the small lower bound for each cluster size, which leads to the larger schedule length than that in the case of δ_{opt}. Thus, we can see that the schedule length in the case that the lower bound for each cluster size is $1.2\delta_{opt}$, $1.5\delta_{opt}$, $2.0\delta_{opt}$, $3.0\delta_{opt}$ (i.e., $y < 1$) is larger than that in the case of δ_{opt}.

Next, let assume the case that the lower bound for each cluster size is smaller than δ_{opt}, i.e., $0.25\delta_{opt}$, $0.5\delta_{opt}$, $0.8\delta_{opt}$. In those cases, communications among tasks are less localized, while it is conceivable that the number of independent tasks in a non-linear cluster is less than the case of δ_{opt}. Thus, the larger CCR becomes, the larger the effect of unlocalized communication on the schedule length becomes. As a result, in cases of $0.25\delta_{opt}$, $0.5\delta_{opt}$ in

figure 3.12, the schedule length tends to be increased. From table 3.7, the number of clusters in which at least one task belongs to seq_R and the number of non-linear clusters is larger or equal to that in the case of δ_{opt}. On the other hand, from figure 3.12, in cases of $0.25\delta_{opt}$, $0.5\delta_{opt}$ the schedule length is larger than that in the case of δ_{opt} even if CCR is 0.1, 0.5. In those two cases, the reasons are that the degree of task parallelism is not retained due to the large number of non-linear clusters and advantages of the communication localization is not obtained. Also, if the case that the lower bound for each cluster size is $0.8\delta_{opt}$, the schedule length is larger than that in the case of δ_{opt}, except cases of figure 3.12 (a) (CCR=7.0) and (b) (CCR=3.0) Especially, in the case of CCR=10.0, the schedule length is larger than that in the case of δ_{opt}. Thus, in the case of $0.8\delta_{opt}$, the larger CCR becomes, the more the communication localization is needed, i.e., it is necessary to make the lower bound larger. From those results, it can be concluded that the schedule length in almost all cases is larger than that in the case of δ_{opt}. If eq.(3.18) is applied to eq.(3.17), eq.(3.17) is monotonically increasing. Thus, if y is increased, the upper bound of $sl_w(G^R_{cls})$ is also increased, thereby $sl_w(G^R_{cls})$ after the task clustering is increased. From results in sec.3.3.8, this fact leads to the increase of both the lower bound and the upper bound of the schedule length. From those characteristics and results in fiure 3.12 and table 3.7, it can be said that δ_{opt} with $y = 1$ at eq.(3.18) is the near-optimal lower bound for each cluster size to minimize ths schedule length. On characteristic of the proposal is to decide the lower bound for each cluster size before each task merging step.

If the lower bound for the "next cluster" to be generated by the next task merging step, in which each task size, each data size and precedence relationships among tasks is considered, is dynamically decided, the better schedule length may be obtained. However, this requires to additional procedures for deriving the lower bound for each task merging step. Such a dynamic approach impose the problem that how to suppress the time complexity of the algorithm, which is one of our future works.

From results describe above, we can obtain a better schedule length by applying δ_{opt}, though it is not always minimized.

3.5.9 Comparison of the running time

We compared the proposed algorithm to other conventional algorithms in order to confirm that the proposal is a practical method or not.

The range of procedures to be measured is from line 0 to 7 in figure 3.5, i.e., procedures for deriving δ_{opt} and $sl_w(G^R_{cls})$ is included in the range. In conventional approaches, CASS-II+LB and DSC+CM are included as comparison targets, because they adopt to derive the critical path length as preprocesses. That is, the running time to be measured is the time duration from the start time of preprocesses to the time that G^R_{cls} is obtained for every approach. Also, as a cluster generation policy, it is the same as that described in sec.3.5.5, i.e., the same number of clusters with CCR being varied. Then by using 100 random DAGs with each task size and each data size being set according to uniform distribution, we compared the averaged running time.

Table 3.8 shows the comparison result in terms of the running time. In this figure, $\varepsilon(G^0_{cls})$ is the value at eq.(3.39). If this value becomes larger, the upper bound of the number of reference to the same cluster (the number of task merging steps for each cluster, i.e., the

Table 3.9: Comparison of running time of each algorithm(Each task and data size is assigned according to random value in normal distribution)
(appears in [22]).

No.	$\|V\|$	α	β	CCR	$\varepsilon(G^0_{cls})$	$\|V^R_{cls}\|$	Algorithm Running Time(ms)			
							A.	B.	C.	D.
1	500	0.1	0.9	0.1	87	104	23	32	31	39
2	500	0.1	0.9	3.0	488	35	31	54	39	55
3	500	0.1	0.9	10.0	876	14	54	86	67	47
4	500	0.9	0.1	0.1	121	114	25	38	39	43
5	500	0.9	0.1	3.0	664	22	34	41	44	49
6	500	0.9	0.1	10.0	1083	13	36	46	51	56
7	1000	0.1	0.9	0.1	103	267	39	94	101	134
8	1000	0.1	0.9	3.0	195	58	79	118	124	169
9	1000	0.1	0.9	10.0	328	36	110	162	187	172
10	1000	0.9	0.1	0.1	41	246	31	85	94	125
11	1000	0.9	0.1	3.0	226	49	71	93	109	156
12	1000	0.9	0.1	10.0	413	23	94	117	111	162

number of selection as $pivot_s$) becomes larger, thereby the running time is also increased. At the proposal (A at table 3.8), the number of clusters is decreased with δ_{opt} being larger. Thus, the fact that $|V^R_{cls}|$ is less corresponds to the increase of the number of task merging steps and $\varepsilon(G^0_{cls})$ becomes larger.

In every cases of $|V|$ is 500 and 1000, the running time becomes lower with the number of clusters being decreased. Also, time complexities of CASS-II and DSC are $O(|E| + |V| \log |V|)$ [29], $O((|V| + |E|) \log |V|)$ [18], while the time complexity of the proposed task clustering is $O(\varepsilon(G^0_{cls})|V|(|V| + |E|))$ as described in sec.3.4.8. CASS-II has the least time complexity. However, the running time of CASS-II+LB becomes larger than that of the proposal, because each cluster is merged by LB after CASS-II has performed. The same reason is applied to DSC, which has the larger running time than that of the proposal. As for LB, the running time is larger because every task is traced for each task merging step. On the other hand, the proposal has the smaller running time than other approaches in every case in table 3.8. From the result in sec. 3.5.5, $sl(G^R_{cls})$ by the proposal can be made lower than that by other approaches with CCR being larger. Thus, it is concluded that the proposal provide the low schedule length with the low running time.

In table 3.8, each task size and data size are assigned according to uniform distribution. Hence, for a cluster generated by R task merging step (i.e., $cls_R(i)$), the number of task merging steps among all approaches required for generating $cls_R(i)$ is almost the same. Thus, it is conceivable that the actual number of task merging steps is not reached to $\varepsilon(G^0_{cls})$.

Next, we compared the running time in the case that a DAG has a deviation on the number of reference to a cluster (the number of task merging steps required for each cluster, i.e., the number of selection for $pivot_s$). In this experiment, let $|V| = 500, 1000$ and each task size and each data size are set according to normal distribution. Table 3.9 the comparison result. In this table, α and β correspond to them at eq.(3.41), while A, B, C, D correspond to them in table 3.8, respectively. In table 3.9, when $\alpha = 0.1$ (No. 1, 2, 3, 7, 8, 9), each task size tend to be deviated to the minimum value. Thus, tasks with small size tend to belong to the same cluster, thereby the long running time is required for update procedures of $S(n^{s+1}_{p'}, p)$

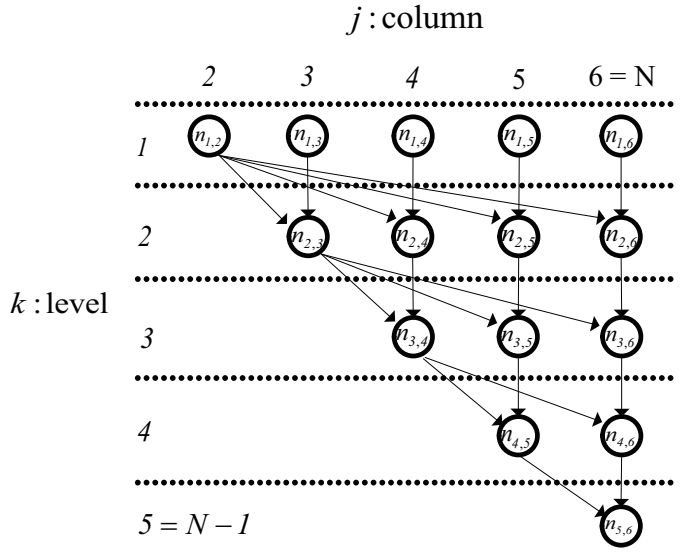


Figure 3.13: An example of Gaussian Elimination DAG structure when $N = 6$ (appears in [22]).

at line 10 in figure 3.8. Thus, in the proposal, the running times at No. 1, 3, 6, 7, 9, 12 are longer than them in table 3.8. However, B, C, and D in table 3.8 have longer running time than the proposal. From results described above, the running time by the proposal is lower than that in other conventional approaches, even if each task size and each data size have a deviation.

3.5.10 Comparison of the degree of effective use of processors in specific applications

CCR is generally derived from each task size (task execution time) and each data size (communication time). That is , CCR depends on the information about the DAG and the execution environment (processor speeds, communication bandwidths and so on). In this experiment, how efficiently each PE is utilized by the proposal in specific DAGs. Those DAGs are Gaussian Elimination DAG [34] and FFT DAG [36], which are used for evaluation of the task scheduling [38,39]. The degree of effective use of processors is defined as the following evaluation formula, by which we compared with other approaches.

$$E(|V_{cls}^R|, Algorithm) = \frac{schedule\ length\ by\ 1\ processor}{|V_{cls}^R| \times (sl(G_{cls}^R)\ by\ the\ Algorithm)}. \tag{3.42}$$

If the value at eq.(3.42) becomes larger, it can be said that each PE is more efficiently utilized. The execution model in this experiment also complies with that defined in sec.3.2.

Table 3.10: Comparison of $E(|V_{cls}^R|, Algorithm)$ in Gaussian Elimination DAG (appears in [22]).

No.	$\|V\|$	$t_p : t_c : \beta$	$\|V_{cls}^R\|$	$E(\|V_{cls}^R\|, Algorithm)$		
				A. Proposal	B. CASS-II+LB	C. DSC+CM
1	1770	1:10:500	9	0.6369	0.4215	0.5232
2	1770	1:20:1000	5	0.7015	0.5624	0.4790
3	1770	1:20:2000	4	0.6655	0.4269	0.3902
4	1770	1:40:2000	4	0.5812	0.5539	0.3152
5	3160	1:10:500	14	0.6097	0.4352	0.5348
6	3160	1:20:1000	11	0.5073	0.3390	0.3626
7	3160	1:20:2000	7	0.6187	0.4975	0.3652
8	3160	1:40:2000	7	0.5157	0.4393	0.2826

Comparison of the degree of effective use of processors with a Gaussian Elimination DAG

In the Gaussian Elimination DAG, the total number of tasks is increased with the order of $O(N^3)$ in the matrix size N by the triple for loop if one assignment statement is handled as one task. It follows that the number of edges is increased, thereby each communication latency becomes large. Thus, we use a kji Gaussian Elimination without pivoting, in which the second inner loop procedure (i.e., **for**$(j = k + 1; j \leq N; j++)$) is handled as one task [34]. Then the structure in the Gaussian Elimination (in the case of $N = 6$) is shown in figure 3.13. In this figure, let denote a task $n_{k,j}^0$, where k ($1 \leq k \leq N-1$) is a level index, and j ($k+1 \leq j \leq N$) is a column index. If the processing time of $n_{k,j}^0$ is defined as $w(n_{k,j}^0)$ and the data transfer time from $n_{k,j}^0$ to $n_{k+1,m}^0 (j+1 \leq m \leq N)$ is defined as $c(e_{(k,j),(k+1,m)}^0)$, each value is independent from j, m as follow [34].

$$w(n_{k,j}^0) = (2(N-k)+1)t_p,\ c(e_{(k,j),(k+1,m)}^0) = \beta + (N-k+1)t_c. \tag{3.43}$$

In eq.(3.43), β is the start-up time (data transfer time required for procedures among PEs [45]) performed before the data transfer among tasks. Also, t_p is the time required for one arithmetic operation, and t_c is the time required for data transfer per one data size unit. In the literature [34], it is shown that the value of m (the value based on the ratio of t_c to t_p [34]) when the highest speed up ratio is theoretically obtained and m when the highest speed up ratio is obtained in the Intel iPSC/860 system are similar. In the literature [35], it is proved that the model, where the data transfer time is linearly behaves with β being considered, is a good approximation to same extent. Thus, in this dissertation we define both each task execution time and data tansfer time as eq.(3.43) using β, t_c, t_p. Though β, t_p, t_c depend on the system to be applied, generally t_c is larger than t_p [35, 45], the communication latency by β has effect on the schedule length [45].

In the literature [34], it is defined that $t_c/t_p \fallingdotseq 18$, $\beta/t_c \fallingdotseq 42.5$ with assuming the cluster environment Intel iPSC/860 of the literature [35]. Also, in the cluster environment by Ncube 6400 in the literature [35], it is defined that $t_c/t_p = 12$, $\beta/t_c \fallingdotseq 41.67$[2]

[2]In the literature [34], "REAL*4 *+*+*" value at table 3 in the literature [35] is assigned to t_p. β is the value at table 6 in the literature [35], while t_c is the value, which is 8 times of the data transfer time for 1 byte at table 6 in the literature [35]. Hence, as for Ncube 6400 we decided values of t_p, t_c, β according to the policy.

In this experiment, we set t_c/t_p and β/t_c in order to accommodate Intel iPSC/860 and Ncube 6400 with assuming those environments. Generally, both t_c and β is larger than t_p, which can be varied with the communication condition. Thus, we set t_c/t_p and β/t_c higher than them in those two environments in the literature [35]. However, at eq.(3.43) $c(e^0_{(k,j),(k+1,m)})$ is much larger than $w(n^0_{k,j})$, i.e., β or t_c is much larger than t_p, the schedule length can not be reduced, in this experiment we set $t_p : t_c : \beta$ as $1 : 10 : 500, 1 : 20 : 1000$, $1 : 20 : 2000, 1 : 40 : 2000$, respectively.

From results in sec.3.5.5, sec.3.5.6, and sec.3.5.9, $sl(G^R_{cls})$ by "D (LB)" is obviously larger than that by other three approaches (the proposal, CASS-II+LB and DSC+CM), and also the running time is larger. Thus, comparison targets are selected as "the proposal," "B. CASS-II+LB," and "C. DSC+CM." Under the condition, we compared eq.(3.42) with the number of PEs being decided by the proposal.

Table 3.10 shows the comparison result of the degree of effective use of processors in cases of $|V| = 1770, 3160$. The column "$t_p : t_c : \beta$" corresponds to the ratio of each value defined at eq.(3.43). In every case, $E(|V^R_{cls}|, Algorithm)$ of the proposal has the highest in all cases. From this result, it is concluded that the proposal provides a high degree of effective use of processors in the environment similar to Intel iPSC/860 and Ncube 6400 in the literature [35].

Comparison of the degree of effective use of processors with a FFT DAG

We compared eq.(3.42) using a FFT DAG. In this experiment, we generated a FFT DAG according to the literature [36, 37]. In this case, the number of tasks is $|V| = N \log N$ [36, 37], where N is the matrix size. In a FFT DAG, arithmetic additions and multiplications of complex numbers are performed and then the result is sent to the immediate successor tasks. In the literature [36], if FFT is executed in a real environment, each task size and each data size is approximated to $w(n^0_k) = c(e^0_{k,l}), n^0_k, n^0_l \in V_0$, respectively. However, various task sizes and data sizes can be considered depending on the execution environment. Similar to the literature [39], in this experiment we generated FFT DAGs with CCR being varied from 0.1 to 10.0. Also, the maximum/minimum ratio in terms of each task size and data size is set to 100, and then 100 DAGs are generated. We compared by the averaged $E(|V^R_{cls}|, Algorithm)$ derived for each DAG. Table 3.11 shows comparison results in cases of $|V| = 2048, 4608$. In both cases of $|V| = 2048, 4608$, the degree of effective use of processors of the proposal is low when CCR=0.1, though higher than other approaches. Also, in the case of $|V| = 2048$, the degree of effective use of processors shifts from the increasing to the decreasing at the boundary as CCR=1.0. However, it is increased when CCR=3.0 or more. On the other hand, in the case of $|V| = 4608$, the degree of effective use of processors shifts from the increasing to the decreasing at the boundary as CCR=3.0, though higher than other approaches. From results described above, in a FFT DAG with CCR is ranged from 0.1 to 10.0, the proposal provides a high degree of effective use of processors.

3.5.11 Discussion

From experimental results obtained from previous sections, a good schedule length can be obtained with the small number of PEs by the proposal, if CCR is ranged from 0.1 to

Table 3.11: Comparison of $E(|V_{cls}^R|, Algorithm)$ in FFT DAG (appears in [22]).

No.	$\|V\|$	CCR	$\|V_{cls}^R\|$	$E(\|V_{cls}^R\|, Algorithm)$		
				A. Proposal	B. CASS-II+LB	C. DSC+CM
1	2048	0.1	275	0.2549	0.1053	0.1222
2	2048	1	129	0.3555	0.1558	0.1649
3	2048	3	79	0.2997	0.2439	0.2503
4	2048	5	62	0.3207	0.2911	0.2473
5	2048	8	44	0.3501	0.3030	0.2406
6	2048	10	32	0.3975	0.2866	0.2245
7	4608	0.1	789	0.2260	0.1026	0.1139
8	4608	1	276	0.2934	0.1450	0.2602
9	4608	3	189	0.3634	0.1746	0.1982
10	4608	5	138	0.3347	0.1462	0.1809
11	4608	8	97	0.3368	0.1654	0.2330
12	4608	10	93	0.3273	0.1513	0.1889

10.0. At first, from the result in sec.3.5.4, it is found that precedence relationships among task in a cluster are retained by the proposal. As a result, $sl_w(G_{cls}^R)$ is reduced by the task clustering, which is lower than that by other approaches with the number of PEs decided by the proposal (from the result in sec.3.5.5). That is, from eq.(3.26), it can be said that the lower bound of the schedule length by the proposal is more reduced than that of other approaches. Thus, if the same scheduling policy is applied to every approach, the lower bound of the schedule length by the proposal is lowest. It is conceivable that this fact can lead to the decrease of the schedule length.

Next, we describe the reason why $sl_w(G_{cls}^S)$ of conventional approaches is higher than that of the proposal. In both CASS-II+LB, DSC+CM, each cluster is merged after the task clustering has been finished. At CASS-II and DSC, each task is scheduled according to the specific scheduling policy. If the schedule length is increased by a task merging step because independent tasks are included in the same cluster, the task merging step is rejected [18,29]. As a result, a cluster having only one task can be obtained in the output DAG, thereby the number of clusters may be large. Thus, for a DAG having many tasks, each cluster may be merged by LB or CM. However, the criterion for selecting a cluster to be merged is the cluster size in LB and CM, and no precedence relationship is considered. Hence, in CASS-II+LB, DSC+CM and LB, the number of tasks in top_s can be increased as figure 3.10. Moreover, from figure 3.9, it can be concluded that the more independent tasks are included in the same cluster, the larger $sl_w(G_{cls}^S)$ becomes.

δ_{opt} derived by eq.(3.22) is the lower bound for each cluster size when the upper bound of $sl_w(G_{cls}^R)$ is minimized. This value is derived by each task size, each data size, and sum of task size on the path in the DAG. Thus, δ_{opt} can be varied with the relationship between each task size and data size. Note that the maximum of the sum of each task size is the lower bound of the schedule length. If this value is larger in the DAG, δ_{opt} takes a large value. This fact means that many communications should be localized as much as possible. As a result, to perform task merging steps with the lower bound can lead to that the schedule length is reached to the lower bound of $sl(G_{cls}^R)$. According to the proposal, the task clustering is performed under the condition that the lower bound for each cluster is δ_{opt}. As a result, as

the result in sec.3.5.6, the comparable $sl(G^R_{cls})$ can be obtained with the small number of PEs by the proposal. This can hold true to a specific DAG.

One objective of the proposal is to decide the number of PEs. Hence, ideally the proposal can be applied to the case that the number of PEs is unknown or unbounded in an identical system, where the number of PEs must be decided before a task scheduling. Also, the proposal can be applied to the case that the schedule length must be minimized with the small number of PEs in a system where unbounded number of identical processors exist.

However, in real cases, it is not feasible that the unbounded number of identical PEs exist. In the proposal, each cluster size to be assigned to a PE is δ_{opt} or more by eq.(3.22). Before the task clustering, the number of PEs to be assigned is defined as follow.

$$\left\lfloor \frac{\sum_{k=1}^{|V|} w(n_k)}{\delta_{opt}} \right\rfloor \tag{3.44}$$

Thus, in the environment described in sec.3.2, if the number of PEs defined at eq.(3.44) is available, the proposal has advantages in terms of effective use of processors.

If we assume a broadband network, e.g., the Internet, processing speeds, communication bandwidths, and communication latencies are heterogeneous. If we try to decide the number of required PEs, it becomes more difficult to derive the lower bound for each cluster, because each task execution time can be varied according to an assigned PE's ability. However, if we can decide the required number of PEs in such heterogeneous distributed systems, every household PCs in the world can be utilized like a grid system. Expanding the proposal in order to apply it to heterogeneous distributed systems, is one of challenging issue.

3.6 Conclusion

In this chapter, a method for deciding the number of identical PEs to minimize the schedule length with the small number of PEs is described. In the proposal, the lower bound for each workload to be assigned to a PE is decided before a task scheduling. Moreover, an indicative value $sl_w(G^s_{cls})$, which has effect on the schedule length, is defined in order to decide the lower bound for each workload before the task clustering. Then the lower bound is derived, by which both the lower bound and the upper bound of the schedule length is proved to be minimized. Then we proposed the task clustering algorithm which performs task merging steps until every workload size exceeds the lower bound. From experimental results by simulatins, it is proved that the proposal can provide the high degree of effective use of processors.

Chapter 4

Task clustering in heterogeneous distributed systems

4.1 Introduction

In the execution model, where several computational resources, i.e., PEs (Processor Elements) collaboratively execute the job with communicating each other, how to minimize the schedule length with suppressing communication latencies is one of critical issues for a task scheduling [12, 16]. Also, if multiple applications (jobs) are processed by the set of PEs in a certain range of network, it is important to minimized the completion time for every job. To achieve the goal, for example it is conceivable that the number of PEs is limited for executing a job with minimizing the schedule length, thereby multiple jobs are executed in parallel.

In the previous chapter, we proposed the method for minimizing the schedule length with the small number of identical PEs in a completely connected network [19]. In this method, the number of required PEs is decided by performing a task clustering [17] under the constraint that each assignment unit (cluster) size is bounded by the lower bound. However, in the real situation each processing speed and each communication bandwidth among PEs are quite different. In such heterogeneous distributed systems, the schedule length can be varied depending on not only each cluster's structure, but also each processing speed and each communication bandwidth. Also, when a task clustering is performed in heterogeneous distributed systems, in general two phases, i.e., task merging steps and selection for each PE to be assigned, are needed in order to decide the mapping between each cluster and each PE [42–44]. However, in conventional approaches, there is no criterion for limiting the number of computational resources. For example, let assume the case that three jobs are executed with a ten PEs. If we assume two cases, i.e., (1) each job is sequentially executed with all PEs, (2) each job is executed with three PEs, the completion time in (2) can be lower than that in (1). This reason depends on the policy for deciding how many clusters should be generated and which PE should be assigned for each cluster. That is, it is important to minimize the schedule length by executing jobs in parallel. Thus, our goal is to decide the PE to be assigned which can contribute on the minimization of the schedule length and to impose a certain constraint to each cluster size with considering communications among

PEs. Such a policy is a fundamental issue to achieve effective use of processors.

In this chapter, we present how to derive the lower bound for each cluster to be assigned to a PE when a task clustering is performed in heterogeneous distributed systems.

Experimental results by simulations shows that the schedule length can be minimized by the decided set of PEs by the task clustering in which each cluster size is bounded by the lower bound.

4.2 Related works

In a distributed environment, where each PE is completely connected, task clustering [17, 18, 29] has been known as one of task scheduling methods. In a task clustering, two or more tasks are merged into one cluster by which communication among them is localized, so that each cluster becomes one assignment unit to a PE. As a result, the number of clusters becomes that of required PEs. On the other hand, if we try to perform a task clustering in heterogeneous distributed systems, the objective is to find an optimal PE assignment, i.e., which PE should be assigned to the cluster generated by a task clustering. Furthermore, since the processing time and the data communication time depend on each assigned PE's performance, each cluster should be generated with taking that issue into account. As related works for task clustering in heterogeneous distributed systems, CHP [42], Triplet [43], and FCS [44] have been known.

CHP [42] firstly assumes that "virtual identical PEs," whose processing speed is the minimum among the given set of PEs. Then CHP performs task clustering to generate a set of clusters. In the processor assignment phase, the cluster which can be scheduled in earliest time is selected, while the PE which has possibility to make the cluster's completion time earliest among other PEs is selected. Then the cluster is assigned to the selected PE. Such a procedure is iterated until every cluster is assigned to a PE. In CHP algorithm, an unassigned PE can be selected as a next assignment target because it has no waiting time. Thus, each cluster is assigned to different PE, so that many PEs are required for execution and Hence CHP can not lead to the effective use of processors.

In Triplet algorithm [43], task groups, each of which consists of three tasks, named as "triplet" according to data size to be transferred among tasks and out degree of each task. Then a cluster is generated by merging two triplets according to its execution time and data transfer time on the fastest PE and the slowest PE. On the other hand, each PE is grouped as a function of its processing speed and communication bandwidth, so that several processor groups are generated. As a final stage, each cluster is assigned to a processor groups according to the processor group's load. The processor assignment policy in Triplet is that one cluster is assigned a processor groups composed of two or more PEs. Thus, such a policy does not match with the concept of effective use of processors.

In FCS algorithm [44], it defines two parameters, i.e., β: total task size to total data size ratio (where task size means that the time unit required to execute one instruction) for each cluster and τ: processing speed to communication bandwidth ratio for each PE. During task merging steps are performed, if β of a cluster exceeds τ of a PE, the cluster is assigned to the PE. As a result, the number of clusters depends on each PE's speed and communication bandwidth. Thus, there is one possibility that "very small cluster" is generated and then

Table 4.1: Parameter definition telated to $sl_w(G^s_{cls})$ (Here $n^s_k \in cls_s(i)$).

Parameter	Definition
$top_s(i)$	$\{n^s_k \mid \forall n^s_l \in pred(n^s_k) s.t., n^s_l \notin cls_s(i)\} \cup \{START\ Tasks \in cls_s(i)\}$.
$in_s(i)$	$\{n^s_k \mid \exists n^s_l \in pred(n^s_k) s.t., n^s_l \notin cls_s(i)\} \cup \{START\ Tasks \in cls_s(i)\}$.
$out_s(i)$	$\{n^s_k \mid \exists n^s_l \in suc(n^s_k) s.t., n^s_l \notin cls_s(i)\} \cup \{END\ Tasks \in cls_s(i)\}$.
$btm_s(i)$	$\{n^s_k \mid \forall n^s_l \in suc(n^s_k), s.t., n^s_l \notin cls_s(i)\} \cup \{END\ Tasks \in cls_s(i)\}$.
$desc(n^s_k, i)$	$\{n^s_l \mid n^s_k \prec n^s_l, n^s_l \in cls_s(i)\} \cup \{n^s_k\}$
$S(n^s_k, i)$	$\sum_{n^s_l \in cls_s(i)} t_p(n^s_l, \alpha_p) - \sum_{n^s_l \in desc(n^s_k, i)} t_p(n^s_l, \alpha_p)$.
$tlevel(n^s_k)$	$\begin{cases} \max\limits_{n^s_l \in pred(n^s_k)} \{tlevel(n^s_l) + t_p(n^s_l, \alpha_p) + t_c(e_{l,k}, \beta_{q,p})\},\ if\ n^s_k \in top_s(i). \\ TL_s(i) + S(n^s_k, i),\ otherwise. \end{cases}$
$TL_s(i)$	$\max\limits_{n^s_k \in top_s(i)} \{tlevel(n^s_k)\}$ ..
$blevel(n^s_k)$	$\max\limits_{n^s_l \in suc(n^s_k), n^s_l \notin cls_s(i)} \left\{ t_p(n^s_k, \alpha_p) + t_c(e^s_{k,l}, \beta_{p,q}) + blevel(n^s_l) \right\}$.
$level(n^s_k)$	$tlevel(n^s_k) + blevel(n^s_k)$.
$BL_s(i)$	$\max\limits_{n^s_k \in out_s(i)} \{S(n^s_k, i) + blevel(n^s_k)\}$.
$LV_s(i)$	$TL_s(i) + BL_s(i) = \max\limits_{n^s_k \in cls_s(i)} \{level(n^s_k)\}$.
ϕ_s	$\{\ldots, < cls_s(i), P_p >, \ldots\}$.
$sl_w(G^s_{cls}, \phi_s)$	$\max\limits_{cls_s(i) \in V^s_{cls}} \{LV_s(i)\}$.

FCS can not match with the concept of effective use of processors.

4.3 Indicative value for the schedule length

In the policy described in the literature [19], it limits the number of PEs by imposing the lower bound for each cluster execution time (sum of each task size divided by the processing speed)[1]. In this chapter, we present how to derive the lower bound for each cluster execution time in order to minimize the schedule length with limiting the number of PEs to some extent in heterogeneous distributed systems.

4.3.1 Indicative value $sl_w(G^s_{cls}, \phi_s)$

The schedule length is decided by each task execution time, each data transfer time, and the execution order for each task. Moreover, whether the data transfer time among tasks exists or not depends on the structure in a cluster. Our proposal in this chapter is to generate a cluster after the lower bound of the cluster has been estimated. The indicative value which

[1]In the literature [19], since the assumed system is the set of identical PEs, every processing speed can be set as 1. Hence, in identical PEs, a cluster size is equal to a cluster execution time. The same thing is applied to each data transfer time and each data size.

Table 4.2: Parameter definitions which are used in analysis on $sl_w(G^s_{cls}, \phi_s)$. $(0 \leq s \leq R)$

Parameter	Definition
p	One path of G^0_{cls}, i.e., $\{n^0_0, n^0_1, n^0_2, \ldots, n^0_k\} \cup \{e^0_{0,1}, e^0_{1,2}, \ldots e^0_{k-1,k}\}$, by which a sequence $< n^0_0, n^0_1, n^0_2, \ldots, n^0_k >$ is constructed, where $e^0_{l-1,l} \in E_0$, n^0_0 is a START task and n^0_k is an END task.
$len(p, \phi_0)$	$\sum_{n^0_k \in p} t_p(n^0_k, \max_{\alpha_i \in \alpha}\{\alpha_i\}) + \sum_{e^0_{k,l} \in p} t_c(c(e^0_{k,l})), \max_{\beta_{i,j} \in \beta}\{\beta_{i,j}\})$
$proc(n^s_k)$	The processor to which n^s_k has been assigned.
cp	$\max_p \left\{ \sum_{n^0_k \in p} w(n^s_k) + \sum_{e^0_{k,l} \in p} c(e^s_{k,l}) \right\}$.
$cp(\phi_s)$	$\max_p \left\{ \sum_{n^s_k \in p} t_p(n^s_k, \alpha_p) + \sum_{e^s_{k,l} \in p} t_c(c(e^s_{k,l}), \beta_{p,q}) \right\}$, where n^s_k, n^s_l are assigned to P_p, P_q, respectively.
cp_w	$\max_{p \in G^0_{cls}} \left\{ \sum_{n^0_k \in p} w(n^0_k) \right\}$.
$cp_w(\phi_s)$	$\max_{p \in G^s_{cls}} \left\{ \sum_{n^s_k \in p} t_p(n^s_k, \alpha_p) \right\}$
$g_{\min}$ [16,24]	$\min_{n^0_k \in V^0_{cls}} \left\{ \frac{\min_{n^0_j \in pred(n^0_k)}\{w(n^0_j)\}}{\max_{n^0_j \in pred(n^0_k)}\{c(e^0_{j,k})\}}, \frac{\min_{n^0_l \in suc(n^0_k)}\{w(n^0_l)\}}{\max_{n^0_l \in suc(n^0_k)}\{c(e^0_{k,l})\}} \right\}$.
$g_{\min}(\phi_s)$	$\min_{n^s_k \in V^s_{cls}} \left\{ \frac{\min_{n^s_j \in pred(n^s_k)}\{t_p(n^s_j \alpha_p)\}}{\max_{n^s_j \in pred(n^s_k)}\{t_c(e^s_{j,k}, \beta_{p,q})\}}, \frac{\min_{n^s_l \in suc(n^s_k)}\{t_p(n^s_l, \alpha_r)\}}{\max_{n^s_l \in suc(n^s_k)}\{t_c(e^s_{k,l}, \beta_{q,r})\}} \right\}$.
$\phi_{s,identical}$	A mapping of clusters to identical processors after s task merging steps.

can be the upper bound of the schedule length is estimated in advance, and then the lower bound for each cluster execution time by which the indicative value can be minimized is derived. Let define the indicative value as $sl_w(G^s_{cls}, \phi_s)$, where ϕ_s is the set of mappings between a cluster to a PE). More specifically, this indicative value is the maximum of the execution path length (the sum of each task execution time and each data transfer time on the execution path), provided that each task is executed as late as possible and every data is arrived at a task (PE).

Also, $sl_w(G^s_{cls}, \phi_s)$ is the value decided after both the each cluster generation is completed and each cluster is assigned to a PE, and the value does not depend on the scheduling policy. Table 4.1 shows definitions for deriving $sl_w(G^s_{cls}, \phi_s)$. In the table, let target PE to be assigned to $cls_s(i), cls_s(j)$ be P_p, P_q, respectively. From $sl_w(G^s_{cls}, \phi_s)$, we can specify the task by which the schedule length can be maximized if the task is execute as late as possible. For example, for n^s_k such that $sl_w(G^s_{cls}, \phi_s) = level(n^s_k)$, the schedule length can be maximized if n^s_k is executed as late as possible.

4.3.2 Relationship between WSL and the lower bound of the schedule length

In an initial DAG, i.e., $G^0_{cls} = (V_0, E_0, V^0_{cls})$, let assume that each task belongs to a different cluster, i.e., every cluster has only one task. From the condition, we analyze the relationship between $sl_w(G^R_{cls}, \phi_R)$ and the schedule length after R task merging steps in heterogeneous

distributed systems. At first, two lemmas are presented in terms of an identical processor system, which is proved in the literature [16].

Lemma 3. In an identical processor system, if cp_w and g_{min} are values according to definitions at table 4.2, we have

$$cp \leq \left(1 + \frac{1}{g_{min}}\right) cp_w. \tag{4.1}$$

■

Lemma 4. In an identical processor system, we have the following relationship.

$$cp_w \leq sl(G^R_{cls}), \tag{4.2}$$

where $sl(G^R_{cls})$ is the schedule length after R task merging steps have been performed in an identical processor system. ■

Moreover, figure 4.1 shows the state after clusters have been generated as our assumption. In this figure, (a) is the initial state, and (b) is the state after five task merging steps have been performed. In (a), each task belongs to an individual cluster. Let assume that the processor P_{max} is virtually assigned to each cluster at the initial state in order to decide both eath task execution time and each data transfer time. Let define the state as ϕ_0, where the processing speed and the communication bandwidth are the maximum of α and β, respectively. By using the state, the following lemma holds.

Lemma 5.

$$cp_w(\phi_0) \leq sl(G^s_{cls}, \phi_s).\blacksquare \tag{4.3}$$

At the state after a cluster mapping is finished in heterogeneous distributed systems, if each task execution time and each data transfer time are assigned to each node and each edge in G^s_{cls}, following corollaries hold by lemma 3, 4, and 5.

Corollary 1.

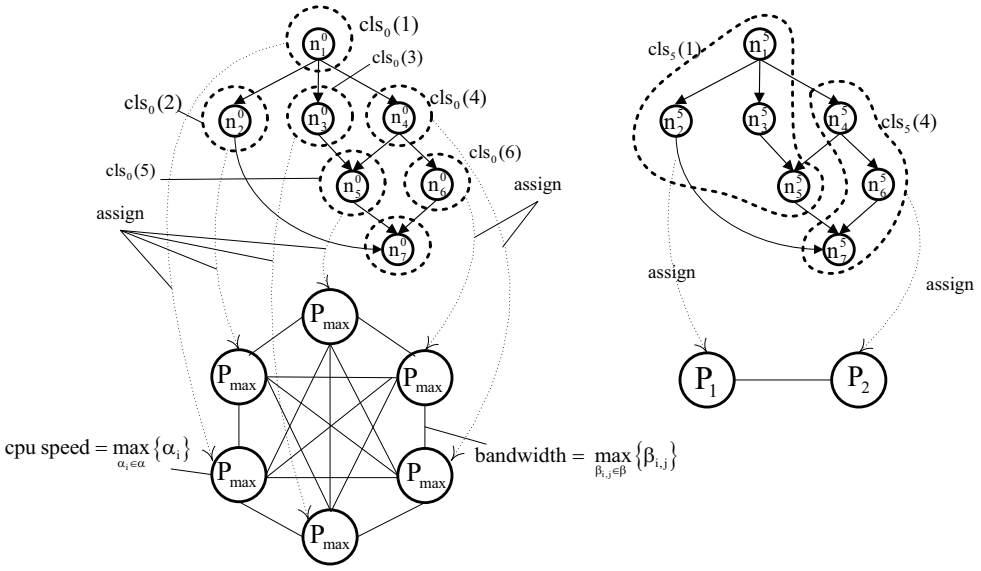
$$cp(\phi_s) \leq \left(1 + \frac{1}{g_{min}(\phi_s)}\right) cp_w(\phi_s),$$

where $g_{min}(\phi_s)$ is the value defined in table 4.2. ■

Corollary 2.

$$cp_w(\phi_0) \leq cp_w(\phi_s) \leq sl(G^s_{cls}, \phi_s).\blacksquare$$

The following definition is derived from corollary 1 and 2.



(a) Initial state (DAG : G^0_{cls}, Assignment : ϕ_0). (b) The state after task clustering (DAG : G^5_{cls}, Assignment : ϕ_5).

Figure 4.1: Assumed condition during cluster generation procedures (appears in [23]).

Thorem 4.1. In heterogeneous distributed systems, let the DAG after R task merging steps have been performed be G^R_{cls}, and assume that every cluster in V^R_{cls} is assigned to a PE in P. At this time, the schedule length is $sl(G^R_{cls}, \phi_R)$. If we define $\Delta sl_{w,up}$ such that $sl_w(G^R_{cls}, \phi_R) - cp(\phi_0) \leq \Delta sl_{w,up}$, the following relationship hold.

$$sl(G^R_{cls}, \phi_R) \geq \frac{sl_w(G^R_{cls}, \phi_R) - \Delta sl_{w,up}}{1 + \frac{1}{g_{min}(\phi_0)}}.$$

Proof 8. From the condition and corollary 1, the following relationship holds.

$$\begin{aligned} & sl_w(G^R_{cls}, \phi_R) - \left(1 + \frac{1}{g_{min}(\phi_0)}\right) cp_w(\phi_0) \\ & \leq \Delta sl_{w,up}. \end{aligned} \tag{4.4}$$

Also, from corollary 2, we have $cp_w(\phi_0) \leq sl(G^R_{cls}, \phi_R)$. By applying this fact into eq.(4.4), we have

$$\begin{aligned} & sl_w(G^R_{cls}, \phi_R) - \left(1 + \frac{1}{g_{min}(\phi_0)}\right) sl(G^R_{cls}, \phi_R) \leq \Delta sl_{w,up} \\ & \Leftrightarrow \\ & sl(G^R_{cls}, \phi_R) \geq \frac{sl_w(G^R_{cls}, \phi_R) - \Delta sl_{w,up}}{1 + \frac{1}{g_{min}(\phi_0)}}. \end{aligned}$$

■

$\Delta sl_{w,up}$ in theorem 4.1 is the upper bound of the increase of the schedule length from the critical path length at the initial state. If each cluster is generated and the mapping ϕ_R is decided, both g_{min} and $\Delta sl_{w,up}$ take fixed values. That is, if $\Delta sl_{w,up}$ is decided, the lower bound of the schedule length can be minimized by generating a cluster such that $sl_w(G^R_{cls}, \phi_R)$ is minimized as much as possible by a task clustering. In sec.4.4.1, we present one example of derivation of $\Delta sl_{w,up}$.

4.3.3 Ralationship between WSL and the upper bound of the schedule length

From definitions in table 4.1, $tlevel(n^s_k)$ is decided based on the data waiting time for each task in $top_s(i)$ (where let $n_k \notin top_s(i)$), thereby $sl_w(G^R_{cls}, \phi_R)$ is also decided based on the data waiting time for each task in $top_s(i)$. On the other hand, the schedule length is decided based on not only tasks in $top_s(i)$, but also the data waiting time for tasks except in $top_s(i)$. Thus, the magnitude relation between $sl_w(G^s_{cls}, \phi_s)$ and the schedule length depends on the magnitude relation between $tlevel(n^s_k)$ and the start time of n^s_k for each task $n^s_k \in cls_s(i)$. If $tlevel(n^s_k)$ is larger than the start time of n^s_k, $sl_w(G^s_{cls}, \phi_s)$ becomes larger than the schedule length. That is, $sl_w(G^s_{cls}, \phi_s)$ becomes the upper bound of the schedule length, then the upper bound of the schedule length can be shorter if $sl_w(G^s_{cls}, \phi_s)$ is small. Even if $tlevel(n^s_k)$ is earlier than the start time of n^s_k, the schedule length can be suppressed because the completion time of its predecessor tasks can be suppressed by the minimization of $sl_w(G^R_{cls}, \phi_R)$. Hence, to make $sl_w(G^s_{cls}, \phi_s)$ smaller leads to the decrease of the schedule length. In a specific case, the following lemma is proved.

Thorem 4.2. *In a heterogeneous distributed sytem, if and only if* $sl(G^s_{cls}, \phi_s) \leq cp(\phi_0) = sl(G^0_{cls}, \phi_0)$, *we have*

$$sl(G^s_{cls}, \phi_s) \leq sl_w(G^s_{cls}, \phi_s) + \zeta - \lambda - \mu, \tag{4.5}$$

where

$$\zeta = \max_{p} \left\{ \sum_{\substack{n_k^0, n_l^0 \in p, \\ n_k^0 \in pred(n_l^0), \\ t_c(e_{k,l}^s, \max_{\beta_{i,j} \in \beta}\{\beta_{i,j}\})=0}} t_c(e_{k,l}^0, \max_{\beta_{i,j} \in \beta}\{\beta_{i,j}\}) \right\}, \tag{4.6}$$

$$\lambda = \min_{p} \left\{ \sum_{\substack{n_k^0 \in p, \\ proc(n_k^s)=p_m}} \left(t_p(n_k^s, \alpha_m) - t_p(n_k^0, \max_{\alpha_i \in \alpha}\{\alpha_i\}) \right) \right\}, \tag{4.7}$$

$$\mu = \min_{p} \left\{ \sum_{\substack{n_k^0, n_l^0 \in p, proc(n_k^s)=P_i, \\ proc(n_l^s)=P_j, \\ n_k^0 \in pred(n_l^0)}} \left(t_c(e_{k,l}^s, \beta_{i,j}) - t_c(e_{k,l}^0, \max_{\beta_{i,j} \in \beta}\{\beta_{i,j}\}) \right) \right\}. \tag{4.8}$$

p and $proc(n_k^0)$ are defined in table 3.2. That is, ζ, λ, μ is derived by scanning every path in the DAG.

Proof 9. *After s task merging steps, there may be both localized edges and not localized edges which compose $sl_w(G_{cls}^s, \phi_s)$. Obviously, we have $sl_w(G_{cls}^0, \phi_0) = cp(\phi_0)$, such edges are not always ones which belongs to $cp(\phi_0)$. Hence the lower bound of $sl_w(G_{cls}^s, \phi_s) - cp(\phi_0)$ can be derived by three factors, i.e., decrease of the data transfer time by localization in one path, increase of the processing time by task merging steps (from ϕ_0 to ϕ_s), and increase of data transfer time for each unlocalized edges (from ϕ_0 to ϕ_s). The localized data transfer time is derived by taking the sum of localized data transfer time for one path. On the other hand, if increase of the processing time is derived by taking the minimum of the sum of increase of task processing time from ϕ_0 to ϕ_s for each path, this value is λ or more. The unlocalized data transfer time is expressed as μ. Then we have*

$$-\zeta + \lambda + \mu \leq sl_w(G_{cls}^s, \phi_s) - cp(\phi_0). \tag{4.9}$$

If $sl(G_{cls}^s, \phi_s) \leq cp(\phi_0) = sl(G_{cls}^0, \phi_0)$, we obtain

$$-\zeta + \lambda + \mu \leq sl_w(G_{cls}^R, \phi_R) - sl(G_{cls}^R, \phi_R) \tag{4.10}$$

$$\Leftrightarrow sl(G_{cls}^R, \phi_R) \leq sl_w(G_{cls}^R, \phi_R) + \zeta - \lambda - \mu. \blacksquare \tag{4.11}$$

4.4 Derivation of the lower bound for each cluster execution time

4.4.1 Assumed situation

In heterogeneous distributed systems, let the set of unmerged tasks after s task merging steps ($1 \leq s \leq R$) be V_{ucls}^s, where each task in V_{ucls}^s is assigned to P_{max}, respectively. Let seq_s

be the set of tasks by which $sl_w(G^s_{cls}, \phi_s)$ is decided, which is similar to table 3.2. $seq^{\prec}_s$ is the path in which every task in seq_s has precedence relationships with other tasks in seq_s.

Here, let assume the situation that the PE P_p is selected as an assignment target before the s-th task merging step. At first, every task in $seq^{\prec}_{s-1}$ is regarded as "unmerged." Then several clusters is virtually generated by temporarily assuming that task merging steps have been performed to tasks in $seq^{\prec}_{s-1}$ in an identical processor system. Let define the lower bound for a cluster execution time in this time as δ. That is, δ is the lower bound for the cluster execution time in the case that the system is temporarily assumed as an identical processor system. By varying δ, the lower bound of the cluster execution time is decided for the set of tasks in $seq^{\prec}_{s-1}$. This is described in sec.4.4.2. Moreover, let define the upper bound of $sl_w(G^s_{cls}, \phi_{s,identical}) - sl_w(G^0_{cls}, \phi_0)$ as $\Delta sl^{s-1}_{w,up}$, where $\phi_{s,identical}$ is the mapping when the PE P_p is assigned to each cluster in $seq^{\prec}_{s-1}$.

That is, $\Delta sl^{s-1}_{w,up}$ is the value derived by estimating the state after the s-th task merging step has been performed after $s-1$ task merging steps[2].

Example 6. *Figure 4.2 shows one example for how to derive $\delta^s_{opt}(P_p)$, where $s = 5$. (a) corresponds to the state after the 4-th (i.e., $(s-1)$-th) task merging step. Four clusters have been generated with their cluster execution times exceeding each lower bound. Further, those clusters have already been assigned to each processor. On the other hand, in (b), it is assumed that all tasks in the path dominating $sl_w(G^4_{cls}, \phi_4)$ (i.e., $sl_w(G^{s-1}_{cls}, \phi_{s-1})$) are merged into several clusters. At the same time, every cluster execution times is assumed to be $\delta^5_{opt}(\alpha, \beta(p)$ or more over the identical processors. This means that in the set of tasks denoted as "Path of $sl_w(G^4_{cls}, \phi_4)$ at (a)," we try to find the lower bound for every cluster and the number of those clusters with virtually assuming the identical processors. (c) is the state after 6 task merging step, where one cluster with size is $\delta^5_{opt}(\alpha, \beta(p)$ has been generated by (b).* ■

4.4.2 Policy for deriving the lower bound for each cluster execution time

Let δ when $\Delta sl^{s-1}_{w,up}$ is minimized be $\delta_{opt}(P_p)$, where δ is a variable. Then assume that one cluster whose execution time is $\delta_{opt}(P_p)$ or more is generated in the set of unmerged tasks in $seq^{\prec}_{s-1}$ after the $s-1$-the task merging step. From the condition described in sec.4.4.1, the following relationship hold.

Corollary 3. *Let $1 \leq s \leq R$. Then if*

$$sl_w(G^s_{cls}, \phi_{s,identical}) - sl_w(G^0_{cls}, \phi_0) \leq \Delta sl^{s-1}_{w,up}$$

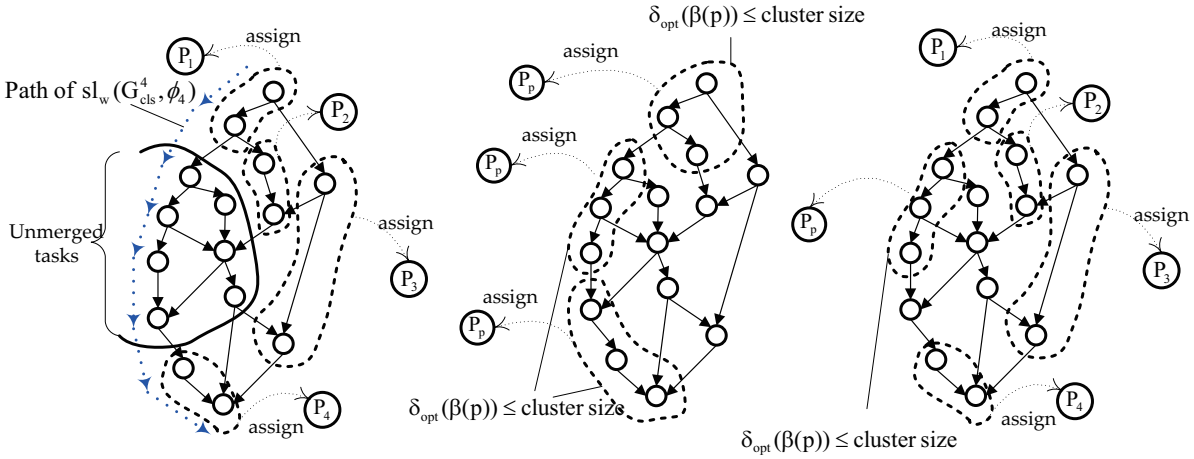
is satisfied and in terms of $\Delta sl^{s-1}_{w,up}$ decided before the s-th task merging step, we have

$$sl_w(G^R_{cls}, \phi_R) - cp(\phi_0) \leq \Delta sl^{s-1}_{w,up},$$

at s-th task merging step,

$$sl(G^R_{cls}, \phi_R) \geq \frac{sl_w(G^R_{cls}, \phi_R) - \Delta sl^{s-1}_{w,up}}{1 + \frac{1}{g_{min}(\phi_0)}}.$$

holds. ■



(a) State after 4 task mergings. (b) Try to minimize $sl_w(G^4_{cls}, \phi_4)$ over the identical network of P_p. (c) State after 6 task mergings.

Figure 4.2: Example of $\delta^s_{opt}(P_p)$ derivation (where $s = 5$) (appears in [23]).

Before each task merging step, e.g., before the s-th task merging step ($1 \leq s \leq R$), $\Delta sl^{s-1}_{w,up}$ takes the fixed value if $\delta^s_{opt}(P_p)$ is decided. Thus, if $sl_w(G^R_{cls}, \phi_R)$ is small under the condition of $\delta^s_{opt}(P_p)$, the lower bound of the schedule length can be small. Also, if the lower bound for each cluster execution time is set as except $\delta^s_{opt}(P_p)$, $\Delta sl^{s-1}_{w,up}$ takes larger value than the case of $\delta^s_{opt}(P_p)$. At this time, from corollary 3 the upper bound of $sl_w(G^R_{cls}, \phi_R)$ can become large. As a result, at corollary 3 the lower bound of $sl(G^R_{cls}, \phi_R)$ become larger, this may not lead to the minimization of the schdule length. Thus, at every task merging step, i.e., at $1 \leq s \leq R$, by deciding $\Delta sl^{s-1}_{w,up}$ under the condition of $\delta^s_{opt}(P_p)$, $\Delta sl^{s-1}_{w,up}$ is minimized.

Note that $seq^{\prec}_s$ is unknown before the s-th task merging step. Thus, one of policy is to minimize $sl_w(G^s_{cls}, \phi_s)$ as much as possible by merging tasks in $seq^{\prec}_{s-1}$. Here, let assume the situation that the cluster $cls_{s-1}(i)$ is generated by the s-th task merging step and thereby the cluster execution time is δ or more. Also, let assume P_p is assigned to $cls_{s-1}(i)$. Then we derive the upper bound of $LV_s(i)$ (defined in table 4.1) of $cls_{s-1}(i)$ after the s-th task merging step. The derivation is conducted by dividing two cases, i.e., the cluster $cls_{s-1}(i)$ is (i) linear ($\Delta L_{lnr}(i)$), or (ii) non-linear $\Delta L_{nlnr}(i)$. Then the maximum number of clusters on a path is derived as N^{max}_s. Let the number of non-linear clusters on the path be y, and let the upper bound of $(N^{max}_s - y)\Delta L_{lnr}(i) + y\Delta L_{nlnr}(i)$ be $\Delta sl^{s-1}_{w,up}$. Finally, let δ when $\Delta sl^{s-1}_{w,up}$ is minimized be $\delta^s_{opt}(P_p)$.

Example 7. Figure 4.3 shows the set of tasks which dominate $sl_w(G^s_{cls}, \phi_s)$. In the figure, (1) corresponds to the case that each cluster is linear, while (2) corresponds to the case that every cluster is non-linear, where dashed lines means the execution order for each task dominating $sl_w(G^s_{cls}, \phi_s)$. At (1), the execution order of each task is unique because every task in a

[2]Since $\Delta sl^{s-1}_{w,up}$ is the value decided after $s-1$-th task merging step, the superscripts -1 is put on it.

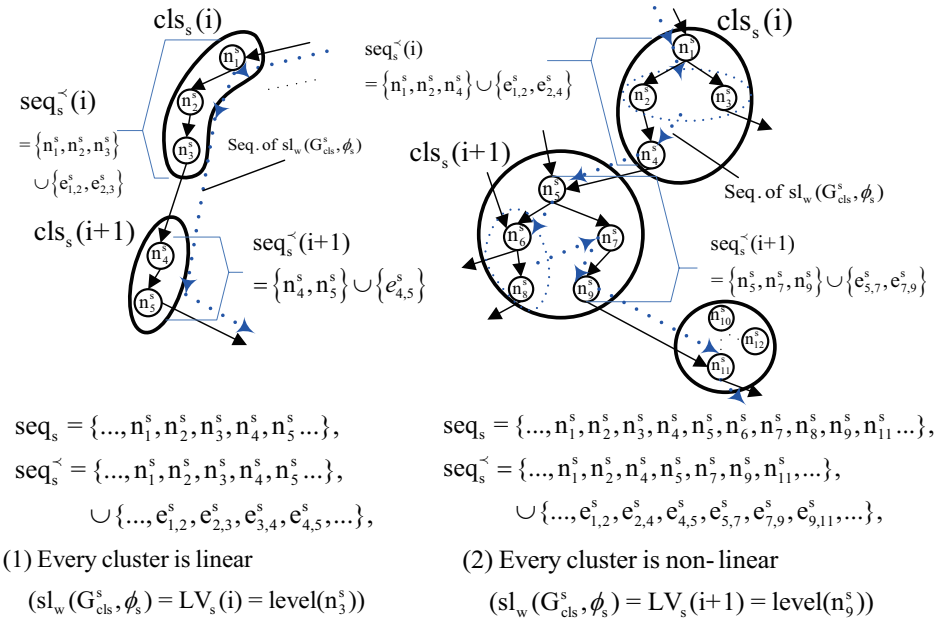


Figure 4.3: Example of an execution route which compose $sl_w(G^s_{cls}, \phi_s)$ (Modified figure 3 from Kanemitsu et. al [22]).

linear cluster has precedence relationships. On the other hand, at (2) an independent task is executed in advance, because each task is executed as late as possible according to the definition of $sl_w(G^s_{cls}, \phi_s)$. For example, in $cls_s(i+1)$ the start time of n^s_7 is delayed by executing n^s_6, n^s_8 before n^s_7. As a result, n^s_9 is executed after every task except n^s_9 in $cls_s(i+1)$ has been completed.

4.4.3 Decision of the lower bound of the cluster execution time

According to the policy described in sec.4.4.1, we describe how to derive $\delta^s_{opt}(P_p)$ after s task merging steps.

(a) The case that a cluster $cls_s(i)$ in which at least one task belong to $seq^{\prec}_{s-1}(i)$ is linear
If we define the processing speed of the PE which is assigned to $cls_s(i)$ as α_p, we have

$$\Delta L_{lnr}(i) = \sum_{n_k^{s-1} \in seq^{\prec}_{s-1}(i)} \left(\frac{w(n_k^{s-1})}{\alpha_p} - \frac{w(n_k^{s-1})}{\max\limits_{\alpha_p \in \alpha} \{\alpha_p\}} \right) - \sum_{e_{k,l}^{s-1} \in seq^{\prec}_{s-1}(i)} \frac{c(e_{k,l}^{s-1})}{\max\limits_{\beta_{p,q} \in \beta} \{\beta_{p,q}\}}. \quad (4.12)$$

(b) The case a cluster $cls_s(i)$ in which at least one task belong to $seq^{\prec}_{s-1}(i)$ is non-linear
Let the lower bound of the cluster execution time be δ (defined in sec.4.4.1). Then if

we define the processing speed of the PE which is assigned to $cls_s(i)$, we have

$$\begin{aligned}\Delta L_{nlnr}(i) &= \left(\delta + \frac{\max\limits_{n_k^{s-1}\in V_s}\left\{w(n_k^{s-1})\right\}}{\alpha_p}\right) - \sum_{n_k^{s-1}\in seq_{s-1}^{\prec}(i)} \frac{w(n_k^{s-1})}{\max\limits_{\alpha_p\in\alpha}\{\alpha_p\}} \\ &\quad - \sum_{e_{k,l}^{s-1}\in seq_{s-1}^{\prec}(i)} \frac{c(e_{k,l}^{s-1})}{\max\limits_{\beta_{p,q}\in\beta}\{\beta_{p,q}\}}. \end{aligned} \tag{4.13}$$

Then let the number of clusters generated by the s-th task merging step on tasks in $seq_{s-1}^{\prec}$ be N_s. If the number of non-linear clusters is y and $sl_w(G_{cls}^s, \phi_s) - sl_w(G_{cls}^0, \phi_0)$ is defined as Δsl_w^s, we have

$$\begin{aligned}\Delta sl_w^s &= sl_w(G_{cls}^s, \phi_s) - sl_w(G_{cls}^0, \phi_0) \\ &\leq \sum_{i=1}^{N_s-y} \Delta L_{lnr}(i) + \sum_{i=N_s-y+1}^{N_s} \Delta L_{nlnr}(i).\end{aligned}$$

By applying eq.(4.12), (4.13), and

$$\min_p\{len(p,\phi_0)\} \leq \sum_{i=1}^{N_s} \sum_{n_k^{s-1}\in seq_{s-1}^{\prec}(i)} \left(\frac{w(n_k^{s-1})}{\max\limits_{\alpha_p\in\alpha}\{\alpha_p\}} + \frac{w(n_k^{s-1})}{\max\limits_{\beta_{p,q}\in\beta}\{\beta_{p,q}\}}\right)$$

to it, we have

$$\begin{aligned}\Delta sl_w^s &\leq \frac{\max\limits_{n_k^0\in V_0}\left\{w(n_k^0)\right\}}{\alpha_p}(N_s - y) + \frac{\max\limits_{e_{k,l}^0\in E_0}\left\{c(e_{k,l}^0)\right\}}{\beta(p)}(N_s - 1) \\ &\quad + y\left(\delta + \frac{\max\limits_{n_k^0\in V_0}\left\{w(n_k^0)\right\}}{\alpha_p}\right) - \min_p\{len(p,\phi_0)\},\end{aligned} \tag{4.14}$$

where

$$\beta(p) = \min\left\{\min_{\substack{1\leq q\leq m,\\ q\neq p}}\{\beta_{p,q}\}, \min_{\substack{1\leq q\leq m,\\ q\neq p}}\{\beta_{q,p}\}\right\}.$$

If the sum of each task execution time in $seq_{s-1}^{\prec}(i)$ is δ or more, the maximum number of clusters generated on a path N_s^{max} is

$$\left\lfloor \frac{\sum\limits_{n_k^{s-1}\in seq_{s-1}^{\prec}} w(n_k^{s-1})}{\alpha_p\delta} \right\rfloor \leq \frac{\sum\limits_{n_k^{s-1}\in seq_{s-1}^{\prec}} w(n_k^{s-1})}{\alpha_p\delta} = N_s^{max}. \tag{4.15}$$

On the other hand, the sum of each task execution time in $seq^{\prec}_{s-1}(i)$ is smaller than δ, eq.(4.14) is monotonically increasing, thereby the upper bound of $sl_w(G^s_{cls}, \phi_s)$ can not be suppressed. Thus, if N^{max}_s is applied to eq.(4.14), we have

$$\begin{aligned}\Delta sl^s_w &\leq \left(\frac{\max\limits_{n^0_k \in V_0} \left\{ w(n^0_k) \right\}}{\alpha_p} + \frac{\max\limits_{e^0_{k,l} \in E_0} \left\{ c(e^0_{k,l}) \right\}}{\beta(p)} \right) \frac{\sum\limits_{n^{s-1}_k \in seq^{\prec}_{s-1}} w(n^{s-1}_k)}{\delta \alpha_p} \\ &\quad + y\delta - \frac{\max\limits_{e^0_{k,l} \in E_0} \left\{ c(e^0_{k,l}) \right\}}{\max\limits_{\beta_{p,q} \in \beta} \{\beta_{p,q}\}} - \min_p \{len(p, \phi_0)\} \\ &= \Delta sl^{s-1}_{w,up}. \end{aligned} \tag{4.16}$$

Also, we have

$$\frac{\partial \Delta sl^{s-1}_{w,up}}{\partial \delta} = 0, \tag{4.17}$$

$$\frac{\partial^2 \Delta sl^{s-1}_{w,up}}{\partial \delta^2} = \frac{2\gamma \sum\limits_{n^{s-1}_k \in seq^{\prec}_{s-1}} w(n^{s-1}_k)}{\alpha_p \delta^3} > 0, \tag{4.18}$$

where

$$\gamma = \frac{\max\limits_{n^0_k \in V_0} \left\{ w(n^0_k) \right\}}{\alpha_p} + \frac{\max\limits_{e^0_{k,l} \in E_0} \left\{ c(e^0_{k,l}) \right\}}{\beta(p)}.$$

$\Delta sl^{s-1}_{w,up}$ takes the local minimum in the case of eq.(4.17), since $\delta > 0$. Though $\Delta sl^{s-1}_{w,up}$ is increased with y being increased, y depends on not only δ, but also the task clustering. Thus, here we set $y = 1$ and from eq.(4.17) we define

$$\delta^s_{opt}(P_p) = \sqrt{\frac{\sum\limits_{n^{s-1}_k \in seq^{\prec}_{s-1}} w(n^{s-1}_k)}{\alpha_p} \left(\frac{\max\limits_{n^{s-1}_k \in V_0} \left\{ w(n^0_k) \right\}}{\alpha_p} + \frac{\max\limits_{e^0_{k,l} \in E_0} \left\{ c(e^0_{k,l}) \right\}}{\beta(p)} \right)}. \tag{4.19}$$

From those results, if the task clustering and the processor mapping which satisfy the condition in corollary 3, $\Delta sl^{s-1}_{w,up}$ in theorem 4.1 exists and can be decreased. After the PE to be assigned is selected, the task merging steps are repeated on the cluster until its execution time is $\delta^s_{opt}(P_p)$ or more, $sl_w(G^R_{cls}, \phi_R)$ can be decreased. This characteristic means that both the lower bound and the upper bound of the schedule length $sl(G^R_{cls}, \phi_R)$ can be decreased, respectively.

4.5 Processor assignment

According to eq.(4.16), since $\delta^s_{opt}(P_p)$ is varied as a function of α_p and $\beta(p)$, it is necessary to decide the PE by assigning $\delta^s_{opt}(P_p)$ to $\Delta sl^{s-1}_{w,up}$ and to find the best combination of the

processing speed and the communication bandwidth. In this case, let the assignment as $\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, \beta(p)))$ as follow.

$$\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, \beta(p))) = \frac{2}{\alpha_p}\sqrt{\sum_{n_k^{s-1} \in seq^{\prec}_{s-1}} w(n_k^{s-1}) \left(\frac{\alpha_p \underset{e^0_{k,l} \in E_0}{max} \left\{ c(e^0_{k,l}) \right\}}{\beta(p)} + \underset{n^0_k \in V_0}{\max} \left\{ w(n^0_k) \right\} \right)} - \frac{\underset{e^0_{k,l} \in E_0}{max} \left\{ c(e^0_{k,l}) \right\}}{\underset{\beta_{p,q} \in \beta}{max} \left\{ \beta_{p,q} \right\}} - \underset{p}{min} \left\{ len(p, \phi_0) \right\}. \quad (4.20)$$

From eq.(4.20), the larger both α_p and $\beta(p)$ become, the smaller $\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, \beta(p)))$ becomes. Hence, the next PE to be selected can be decided by comparing for each $\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, \beta(p)))$, then the PE which has minimum value of $\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, \beta(p)))$ should be selected as the next assignment target.

4.5.1 Characteristics of the next PE

In this section we analyze characteristics of the PE to be assigned. We present relationship between the processing speed and the communication bandwidth of the PE. Then by expressing $\beta(p)/\alpha_p$ as k, $\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, \beta(p)))$ is defined as

$$\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, k\alpha_p)) = \frac{2}{\alpha_p}\sqrt{\sum_{n_k^{s-1} \in seq^{\prec}_{s-1}} w(n_k^{s-1}) \left(\frac{\underset{e^0_{k,l} \in E_0}{max} \left\{ c(e^0_{k,l}) \right\}}{k} + \underset{n^0_k \in V_0}{\max} \left\{ w(n^0_k) \right\} \right)} - \frac{\underset{e^0_{k,l} \in E_0}{max} \left\{ c(e^0_{k,l}) \right\}}{\underset{\beta_{p,q} \in \beta}{max} \left\{ \beta_{p,q} \right\}} - \underset{p}{min} \left\{ len(p, \phi_0) \right\}. \quad (4.21)$$

From eq.(4.21), obviously k should be large in order to minimize $\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, k\alpha_p))$ under the condition that α_p is not varied. Thus, the variation of $\beta(p)$ has more critical impact on $\Delta sl^{s-1}_{w,up}$ than that of α_p. Hence, even if a PE's processing speed is not so high, its communication bandwidth can precede for selecting the next PE. This characteristic is consistent with the fact that communications among PEs can be a bottleneck in the schedule length in distributed systems. Generally, the data transfer time between PEs has larger impact on the schedule length than the processing time, especially for data-intensive DAG.

4.5.2 Overall procedures

The overall procedure for processor assignment consists of three phases: (i) derive the lower bound for each cluster execution time as eq.(4.19), (ii) decide the PE to be assigned, which minimize eq.(4.20). Then (iii) merge several task into a cluster until its size exceeds the lower bound derived in (i). Figure 4.4 shows the whole procedures including those three

INPUT: G^0_{cls}
OUTPUT: G^R_{cls}
Set the mapping state ϕ_0 (each task is single cluster);
Define UEX_s as the set of unmerged clusters;
Define RDY_s as the set of candidates as $pivot_s$;
$s \leftarrow 0$;
1. **WHILE** $UEX_s \neq \emptyset$ **DO**
2. Derive $\delta^{s+1}_{opt}(P_p)$ by eq.(4.19);
3. Find P_p s.t.,$\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(P_p)) = \min_{P_i \in unassigned\ PEs} \{\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(P_i))\}$;
4. $pivot_s \leftarrow getPivot(RDY_s)$;
5. **WHILE** size of $pivot_s < \delta^{s+1}_{opt}(P_p)$ **DO**
6. $target_s \leftarrow getTarget(pivot_s)$;
7. $pivot_{s+1} \leftarrow merge(pivot_s, target_s)$ and update RDY_s as RDY_{s+1};
8. $s \leftarrow s + 1$;
9. **ENDWHILE**
10. Assign P_p to $pivot_s$;
11. **ENDWHILE**

Figure 4.4: Overall procedures for the processor assignment (appears in [23]).

phases. At first, the mapping state ϕ_0 is applied to the input DAG, G^0_{cls}, where every task is virtually assigned to a PE P_{max} having the maximum processing speed and the maximum communication bandwidth. Objectives of the procedures presented in figure 4.4 are as follows.

A. make every cluster execution time exceeds the lower bound.

B. Minimize $sl_w(G^R_{cls}, \phi_R)$, where R is the number of task merging steps taken to satisfy the objective A.

Our approach to satisfy the objective A is to check whether every cluster execution time exceeds the lower bound or not. Thus, UEX_s, which is the set of clusters whose size is under the lower bound, is defined. If every cluster execution time exceeds the lower bound, the procedure is finished. As for the objective B, it complies with theorem 1 and 2, i.e., minimizing $sl_w(G^R_{cls})$ can lead to minimizing the schedule length after R task merging steps. To achieve this objective, our approach is to select two clusters for each task merging by which $sl_w(G^R_{cls})$ is minimized. Thus, At a task merging step (let the step as s-th), $sl_w(G^s_{cls})$ should be minimized. In the procedure in figure 4.4, which contains the set of clusters which may be selected for the $(s+1)$-th task merging step.

4.5.3 Processor selection phase

Before the cluster generation, the processor to be assignment should be selected. Thus, at line 2 in figure 4.4 $\delta^{s+1}_{opt}(P_p)$ is defined before the $(s+1)$-th task merging step. Since $\delta^{s+1}_{opt}(P_p)$ is a function of the processing speed and the communication bandwidth, for each unassigned PE its α_p and $\beta(p)$ are assigned to $\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, \beta(p)))$ in order to decide the next PE to be assigned. If the PE which minimizes $\Delta sl^{s-1}_{w,up}(\delta^s_{opt}(\alpha_p, \beta(p)))$ is found, it is selected as P_p at line 3.

4.5.4 Cluster selection phase

At line 4 in figure 4.4, before $(s+1)$-th task merging step, the cluster dominating $sl_w(G^s_{cls})$ is selected as $pivot_s$ from RDY_s . This is based on the heuristic that reducing $sl_w(G^s_{cls})$ can lead to the reduction of $sl_w(G^{s+1}_{cls})$. Hence, once $pivot_s$ is selected, the other cluster merged with $pivot_s$ should dominate $sl_w(G^{s+1}_{cls})$. Such a cluster is named as $target_s$. At line 5-9 in figure 4.4, this loop is repeated until the size of $pivot_s < \delta^s_{opt}(P_p)$. During the loop, at first $target_s$ is selected and then $pivot_s$ and $target_s$ are merged into the new cluster, i.e., $pivot_{s+1}$ (line 7). In this merging, every task in $target_s$ comes to belong to $pivot_{s+1}$. Then RDY_s is updated to become RDY_{s+1}, which means that the new clusters for the next task merging belongs to RDY_{s+1}, and both $pivot_{s+1}$ and $target_s$ are removed from RDY_{s+1} if they belong to RDY_{s+1}. If the size of $pivot_s$ exceeds $\delta^s_{opt}(P_p)$, the procedure returns to the line 1.

4.5.5 Processor assignment phase

After the next processor, P_p has been decided and the size of $pivot_s$ exceeds $\delta^{s+1}_{opt}(P_p)$, P_p is assigned to $pivot_s$. Then $pivot_s$ is removed from RDY_s. If no cluster exists in UEX_s, the procedure ends, and every task in every processor (cluster) is scheduled by a task scheduling policy to derive the schedule length. Since the proposal in this pa-per focuses on only the processor assignment, we do not mention about task scheduling policy. After the procedure at table 4.4 finished, any task scheduling policy can be applied, because every task has been assigned to a processor at a task scheduling such as list scheduling heuristics [28].

4.6 Experimental comparison

4.6.1 Objective

We conducted the experimental comparison in terms of three points as follows.

(i) To confirm that the schedule length is minimized if $sl_w(G^s_{cls}, \phi_s)$ is minimized, provided that a cluster execution time is $\delta^s_{opt}(P_p)$ or more.

(ii) To confirm that the lower bound for the cluster execution time as $\delta^s_{opt}(P_p)$, by which the upper bound of $sl_w(G^R_{cls}, \phi_R)$ is minimized, is optimal or not.

(iii) Advantages of the processor assignment.

4.6.2 Simulation environment

We describe the experimental environment described at (i) in sec. 4.6.1. In this experiment, we generated random DAGs and FFT DAGs as a realistic job. As for a random DAG, we generated it based on CCR (Communication to Computation Ratio) [16,41]. In this chapter, CCR is the ratio of the average data size to the average task size, which takes from 0.1 to 10.0. In the DAG, the number of tasks is set to 1000, the out degree for each task is from 1 to 5, the ratio of the maximum to the minimum of both data size and task size is 100, respectively. In this experiment at (i) in sec.4.6.1, we compared the degree of contribution of $\delta^s_{opt}(P_p)$ to the minimization of the schedule length. If the ratio of processing speed among

PE is too large, the variation of the schedule length becomes too large due to the difference among assigned PEs, thereby the effect of $\delta^s_{opt}(P_p)$ may not be not obtained. Thus, we set the ratio of the processing speed to the communication bandwidth from 5 to 10. Then the averaged value in 100 tries is used for the comparison.

As for the generation policy for FFT DAG, we set CCR from 0.1 to 10.0, which is similar to that in the literature [39]. Also, the number of tasks is set as 2048 or 4096, the ratio in terms of the processing speed and the communication bandwidth is similar to the case of random DAGs.

In the second point in sec.4.6.1, we generated random DAGs. The condition for the generation is similar to (i) in sec.4.6.1. As for the lower bound of every cluster execution time, we set $0.25\delta^s_{opt}(P_p)$, $0.5\delta^s_{opt}(P_p)$, $\delta^s_{opt}(P_p)$, $1.5\delta^s_{opt}(P_p)$, $2\delta^s_{opt}(P_p)$, $3\delta^s_{opt}(P_p)$ in order to compare the schedule length.

The experimental environment is , JRE1.6.0_03, OS is Windows XP SP3, CPU is Intel Core2 Duo 2.66GHz, and the memory size is 2.0GB.

4.6.3 Procedures

Procedure at (i) in sec.4.6.1

Procedures for implementing the task clustering and the processor assignment are presented. At the case A, the lower bound of the cluster execution time with considering the capability in the PE, while the lower bound is decided without the capability in the case B.

Case A

A-1 Each PE is assigned a cluster according to CHP [42], where the schedule length is derived by assuming that the many PEs having the smallest processing speed exist in "virtual identical processor system."

A-2 Specify $seq^{\prec}_{s-1}$ and then derive $\delta^s_{opt}(P_p)$ for P_p selected in A-1.

A-3 A cluster is generated from the set of task in $seq^{\prec}_{s-1}$ such that the cluster execution time becomes $\delta^s_{opt}(P_p)$ or more. Then the PE P_p is assigned to the generated cluster. The task merging policy is similar to that described in the literature [19], in which $sl_w(G^s_{cls}, \phi_s)$ is minimized as much as possible with considering the mapping state ϕ_s.

A-4 By repeating from A-1 to A-3, the overall procedures is completed when every cluster is assigned to a PE.

Case B

B-1 Generate the set of clusters whose number is equal to that in the case A and whose size is δ_{opt} or more with assuming an identical processor system. The generation policy is the same as that in the literature [19]. If the number of generated clusters is larger than that in the case A, each cluster is merged by Load Balancing algorithm [26], thereby the number of cluster becomes is the same as case A.

B-2 For each cluster, a PE is assigned according to the processor selection policy of CHP. The set of PEs is the same as that in the case A.

Table 4.3: Comparison results in random DAG (♯ of tasks : 1000).

No.	α	β	CCR	$\lvert V_{cls}^R \rvert$	$sl_w(G_{cls}^R, \phi_R)$ ratio		$sl(G_{cls}^R, \phi_R)$ ratio	
					A	B	A	B
1	5	5	0.1	168	1.000	1.126	1.000	1.038
2			1.0	56	1.000	1.213	1.000	1.021
3			5.0	34	1.000	1.384	1.000	1.098
4			10.0	19	1.000	1.411	1.000	1.245
5	5	10	0.1	160	1.000	1.187	1.000	1.092
6			1.0	49	1.000	1.164	1.000	1.156
7			5.0	30	1.000	1.210	1.000	1.242
8			10.0	16	1.000	1.412	1.000	1.199
9	10	5	0.1	150	1.000	1.032	1.000	1.002
10			1.0	47	1.000	1.095	1.000	1.101
11			5.0	41	1.000	1.271	1.000	1.123
12			10.0	26	1.000	1.209	1.000	1.219
13	10	10	0.1	187	1.000	1.197	1.000	1.098
14			1.0	67	1.000	1.211	1.000	1.160
15			5.0	44	1.000	1.280	1.000	1.287
16			10.0	28	1.000	1.367	1.000	1.302

B-3 The procedure is completed when every cluster is assigned a PE.

In both case A and B, the schedule length is derived by RCP scheduling [28] after every cluster is assigned to a PE [3].

Procedure at (ii) in sec.4.6.1

In this experiment, the task clustering and the processor assignment by the case A in sec.4.6.3 is performed to the same DAG in every approaches. Only the lower bound for each cluster execution time is different. 100 DAGs are generated and then we compared by the average schedule length.

Procedure at (iii) in sec.4.6.1

In this experiment, we generated 100 random DAGs and averaged each $sl_w(G_{cls}^R, \phi_R)$ and the schedule length. Then we compared both averaged values among three approaches as follows.

A. Our proposal in this dissertation, i.e., each processor is assigned according to the policy of figure 4.4. Then theDAG is scheduled by RCP scheduling [28]

B. A Conventional processor assignment is performed and the lower bound for each cluster execution time is decided according to the assigned PE. In this case, PE, P_p , is assigned according to CHP [42], Then $\delta_{opt}^s(P_p)$ is derived. Finally, the task clustering

[3]RCP scheduling described in the literature [28] is assumed to be applied to an identical processor system. However, this scheduling algorithm can be applied to heterogeneous distributed systems because each task execution time and each data transfer time are known after the processor mapping has been completed. Also, in this dissertation we do not use a task scheduling for heterogeneous distributed system such as HEFT [39] because such an algorithm includes the processor mapping. Otherwise, we can not do the accurate comparison with the fixed processor mapping policy.

Table 4.4: Comparison results in FFT DAG (♯ of tasks : 2048).

No.	α	β	CCR	$\lvert V_{cls}^R \rvert$	$sl_w(G_{cls}^R, \phi_R)$ ratio		$sl(G_{cls}^R, \phi_R)$ ratio	
					A	B	A	B
1	5	5	0.1	281	1.000	1.341	1.000	1.018
2			1.0	135	1.000	1.218	1.000	1.089
3			5.0	58	1.000	1.281	1.000	1.175
4			10.0	35	1.000	1.398	1.000	1.219
5	5	10	0.1	274	1.000	1.201	1.000	1.031
6			1.0	126	1.000	1.548	1.000	1.087
7			5.0	55	1.000	1.288	1.000	1.124
8			10.0	31	1.000	1.499	1.000	1.217
9	10	5	0.1	252	1.000	1.182	1.000	1.066
10			1.0	119	1.000	1.211	1.000	1.092
11			5.0	48	1.000	1.298	1.000	1.132
12			10.0	30	1.000	1.365	1.000	1.241
13	10	10	0.1	291	1.000	1.176	1.000	1.044
14			1.0	141	1.000	1.362	1.000	1.124
15			5.0	64	1.000	1.277	1.000	1.173
16			10.0	37	1.000	1.386	1.000	1.279

is performed until every cluster execution time exceeds each lower bound derived for each task merging step.

C. Conventional processor assignment by CHP [42] and not changing the lower bound for cluster execution time, i.e., every cluster has the same lower bound.

4.6.4 Comparison result in random DAGs

Table 4.3 shows the comparison result. In the table, α and β correspond to the ratios of the maximum to the minimum value of the processing speed and the communication bandwidth, respectively. $|V_{cls}^R|$ is the number of generated clusters (the number of required PEs), while $sl_w(G_{cls}^R, \phi_R)$ ratio means the one of $sl_w(G_{cls}^R, \phi_R)$ of the case B to that of the case A. $sl(G_{cls}^R, \phi_R)$ ratio means the one of $sl(G_{cls}^R, \phi_R)$ of the case B to that of the case A. From results in 4.3, it is observed that both $sl_w(G_{cls}^R, \phi_R)$ ratio and $sl(G_{cls}^R, \phi_R)$ ratio are increase with CCR being larger as a whole.

When α is fixed and β is varied, e.g., relationships between "No. 1-4 and No. 5-8," and "No. 9-12 and No. 13-16," it is observed that the schedule length in the case A is the smaller in $\beta = 10$ than the case of $\beta = 5$. Since the processor assignment is based on CHP [42], it is common that an unlocalized data is transferred on a large communication link among case A and B. Thus, one of reason for the result is that more appropriate cluster execution time is decided for each PE in the case A than the case B. From this result, it is concluded that it is more important to decide the lower bound for each cluster with considering the PE's capability in the system where CCR of the DAG is high and there is great variability among communication bandwidths.

Table 4.5: Comparison results in FFT DAG (♯ of tasks : 4608).

No.	α	β	CCR	$\lvert V^R_{cls}\rvert$	$sl_w(G^R_{cls}, \phi_R)$		$sl(G^R_{cls}, \phi_R)$	
					A	B	A	B
1	5	5	0.1	701	1.000	1.215	1.000	1.071
2			1.0	288	1.000	1.181	1.000	1.127
3			5.0	155	1.000	1.293	1.000	1.094
4			10.0	88	1.000	1.334	1.000	1.192
5	5	10	0.1	692	1.000	1.112	1.000	1.080
6			1.0	271	1.000	1.244	1.000	1.107
7			5.0	149	1.000	1.301	1.000	1.131
8			10.0	81	1.000	1.311	1.000	1.120
9	10	5	0.1	688	1.000	1.139	1.000	1.097
10			1.0	269	1.000	1.201	1.000	1.072
11			5.0	144	1.000	1.212	1.000	1.172
12			10.0	77	1.000	1.339	1.000	1.207
13	10	10	0.1	734	1.000	1.089	1.000	1.098
14			1.0	298	1.000	1.226	1.000	1.136
15			5.0	159	1.000	1.249	1.000	1.127
16			10.0	92	1.000	1.368	1.000	1.183

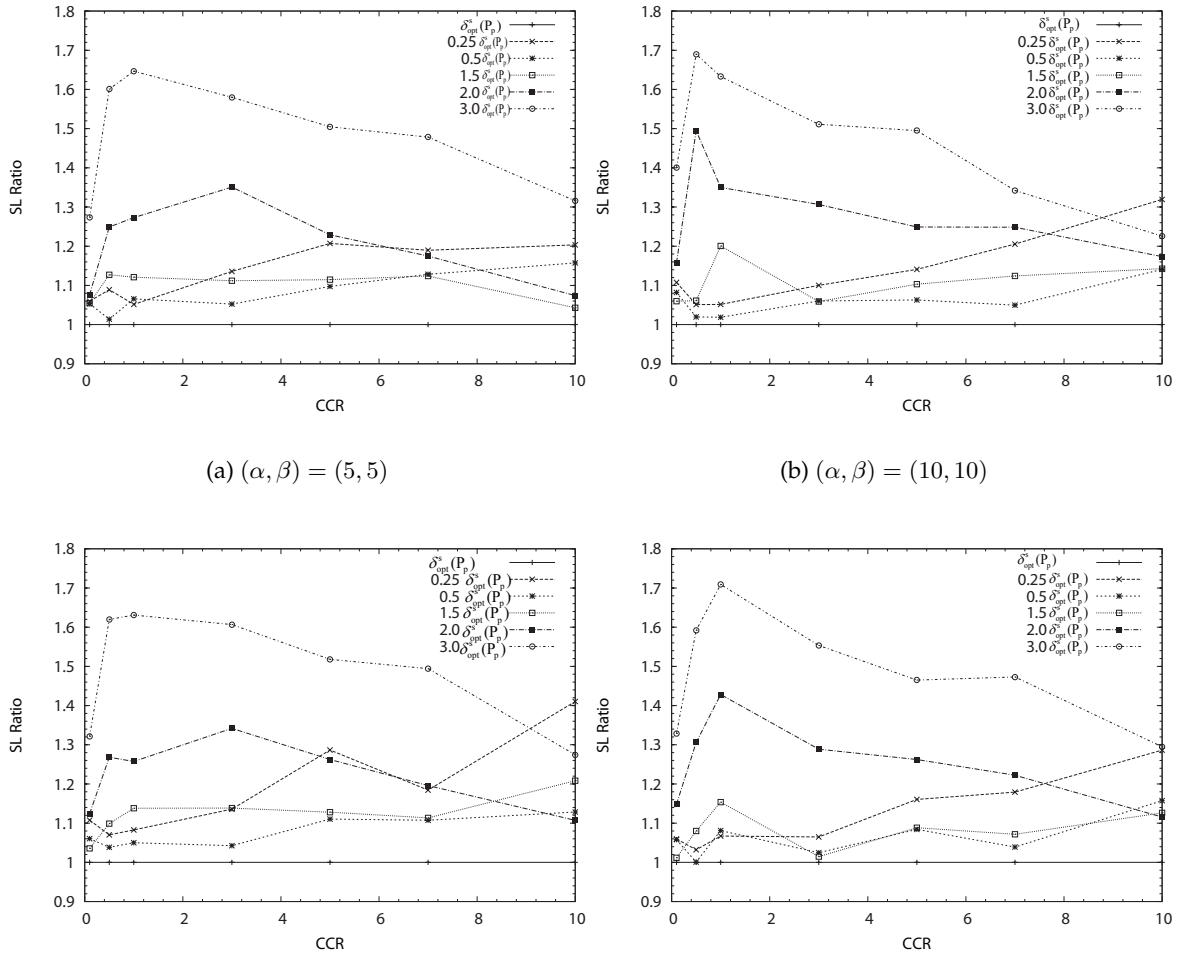
4.6.5 Comparison result in FFT DAGs

We conducted the experiment in terms of (i) in sec.4.6.1 with FFT DAG. Table 4.4 shows the comparison result. Similar to the result in the case of random DAG, in this experiment the difference of the schedule length becomes larger with CCR being larger, and it is found that $sl_w(G^R_{cls}, \phi_R)$ is case A is the lowest. Also, we conducted the experiment when the number of tasks is 4608 and then the similar result is obtained. From those results described above, the lower bound for each cluster execution time is derived by the case A is applicable in a realistic job.

4.6.6 Optimality of the lower bound for each cluster execution time

We compared the schedule length with changing only the lower bound for each cluster execution time, under the condition that both approaches adopt the task clustering and the processor assignment by the case A. Figure 4.5 shows comparison results. In those figure, the x-axis corresponds to CCR (from 0.1 to 10.0), and y-axis corresponds to the ratio of the schedule length when the schedule length by setting the lower bound, i.e., $\delta^s_{opt}(P_p)$ for each cluster as 1.0.

In every figure, the ratio becomes larger with CCR being larger in cases of $0.25\delta^s_{opt}(P_p)$ and $0.5\delta^s_{opt}(P_p)$. This is because that the effect of each data transfer time on the schedule length becomes large due to the small lower bound for each cluster execution time, while each data size becomes large with CCR being larger. Also, in cases of $1.5\delta^s_{opt}(P_p)$, $2.0\delta^s_{opt}(P_p)$, $3.0\delta^s_{opt}(P_p)$, the ratio becomes the largest when CCR takes a certain value, and then the ratio is decreasing when CCR becomes larger then a certain value. This is because a large data communication is localized with CCR being larger due to the larger lower bound for each cluster execution time. However, the schedule length is still larger than the case of $\delta^s_{opt}(P_p)$ $\delta^s_{opt}(P_p)$ can be said to be a near-optimal value from the experiment.



(a) $(\alpha, \beta) = (5, 5)$

(b) $(\alpha, \beta) = (10, 10)$

(c) $(\alpha, \beta) = (5, 10)$

(d) $(\alpha, \beta) = (10, 5)$

Figure 4.5: Optimality of $\delta^s_{opt}(P_p)$

4.6.7 Comparison in tems of processor assignment

Table 4.6 and 4.7 show comparison results in a random DAG and a FFT DAG among three approaches. In both tables, α and β mean the maximum to minimum ratio of the processing speed and the communication bandwidth, respectively. The larger value means high degree of heterogeneity. $|V^R_{cls}|$ is the number of clusters in the case of A. To equalize the number in every approach, in B and C cluster merging steps are performed by LB [26]. "$sl_w(G^R_{cls}, \phi_R)$" corresponds to the ratio of $sl_w(G^R_{cls}, \phi_R)$ values to that of A. Similarly, "$sl(G^R_{cls}, \phi_R)$ " corresponds to the schedule length ratio to that of A. Thus, a value over 1 means that the schedule length is worse than A.

In table 4.6, in every try, our proposal (A) has smaller values in $sl_w(G^R_{cls}, \phi_R)$ and $sl(G^R_{cls}, \phi_R)$ than other approaches. Although it is not found that the correlation in varying the heterogeneity and the schedule length, the correlation in CCR is found. The larger CCR becomes,

Table 4.6: Comparison in terms of processor assignment in random DAGs (♯ of tasks = 1000) (appears in [23]).

No.	α	β	CCR	$\lvert V_{cls}^R \rvert$	$sl_w(G_{cls}^R, \phi_R)$			$sl(G_{cls}^R, \phi_R)$		
					A	B	C	A	B	C
1	5	5	0.1	161	1.000	1.005	1.052	1.000	1.024	1.051
2			1.0	62	1.000	1.012	1.099	1.000	1.063	1.077
3			5.0	47	1.000	1.008	1.143	1.000	1.080	1.138
4			10.0	17	1.000	1.104	1.132	1.000	1.132	1.171
5	5	10	0.1	159	1.000	1.002	1.035	1.000	1.012	1.069
6			1.0	51	1.000	1.051	1.181	1.000	1.077	1.082
7			5.0	28	1.000	1.042	1.124	1.000	1.063	1.124
8			10.0	17	1.000	1.132	1.215	1.000	1.092	1.167
9	10	5	0.1	146	1.000	1.009	1.034	1.000	1.015	1.072
10			1.0	49	1.000	1.018	1.067	1.000	1.056	1.061
11			5.0	38	1.000	1.113	1.133	1.000	1.121	1.132
12			10.0	26	1.000	1.119	1.171	1.000	1.126	1.177
13	10	10	0.1	183	1.000	1.002	1.081	1.000	1.002	1.041
14			1.0	65	1.000	1.108	1.133	1.000	1.042	1.089
15			5.0	43	1.000	1.112	1.177	1.000	1.098	1.075
16			10.0	28	1.000	1.162	1.248	1.000	1.108	1.137

the worse both $sl_w(G_{cls}^R, \phi_R)$ and $sl(G_{cls}^R, \phi_R)$ of C become. This is because a DAG with high CCR requires huge data communication among PEs. Thus there may be some small clusters despite they are assigned to PEs having wide communication bandwidths. Hence, at least the lower bound for each cluster execution time should be adjusted according to the assigned PE's capability. As for comparison between A and B, it can be said that the proposed processor assignment policy has better impact on the schedule length in a DAG with high CCR.

In table 4.7, similar results as table 4.6 are obtained. From comparison results in table 4.6 and 4.7, it is concluded that the proposed processor assignment is superior to other conventional processor assignment methods.

4.6.8 Discussion

From comparison results conducted in previous sections, it is important to decide the lower bound for each cluster execution time according to each PE's capability in the system where variation of each communication bandwidth is large. Before each task merging step, e.g., before the s-th task merging step, it is conceivable that deciding $\delta_{opt}^s(P_p)$ to satisfy the condition in corollary 3 leads to the reduction of the schedule length. By specifying However, if the both the maximum of task size and data size are specified from $seq_{s-1}^{\prec}$, the better lower bound may be obtained, thereby the schedule length can be more reduced. This issue is one of our future works.

From theorem 4.1 and 4.2, it can theoretically be seen that an approach for minimizing $sl_w(G_{cls}^R, \phi_R)$ has good impact on the schedule length. However, since $\delta_{opt}^s(P_p)$ is the lower bound when the upper bound of $sl_w(G_{cls}^R, \phi_R)$ can be minimized, the lower bound for each cluster execution time is "near-optimal" value. Thus, it is necessary to confirm by experiments advantages of the combination of "$sl_w(G_{cls}^R, \phi_R)$ minimization" and "adjusting the lower bound according to the assigned PE's capability." From comparison results presented

Table 4.7: Comparison in terms of processor assignment in FFT DAGs (♯ of tasks = 2048) (appears in [23]).

No.	α	β	CCR	$\lvert V_{cls}^R \rvert$	$sl_w(G_{cls}^R, \phi_R)$			$sl(G_{cls}^R, \phi_R)$		
					A	B	C	A	B	C
1	5	5	0.1	277	1.000	1.009	1.077	1.000	1.003	1.023
2			1.0	131	1.000	1.007	1.156	1.000	1.032	1.041
3			5.0	61	1.000	1.018	1.241	1.000	1.044	1.077
4			10.0	36	1.000	1.054	1.351	1.000	1.071	1.109
5	5	10	0.1	273	1.000	1.008	1.062	1.000	1.005	1.021
6			1.0	129	1.000	1.132	1.142	1.000	1.021	1.058
7			5.0	51	1.000	1.073	1.122	1.000	1.077	1.104
8			10.0	29	1.000	1.149	1.281	1.000	1.074	1.122
9	10	5	0.1	252	1.000	1.003	1.221	1.000	1.002	1.049
10			1.0	118	1.000	1.098	1.091	1.000	1.039	1.081
11			5.0	45	1.000	1.117	1.155	1.000	1.091	1.133
12			10.0	30	1.000	1.083	1.182	1.000	1.104	1.176
13	10	10	0.1	290	1.000	1.006	1.032	1.000	1.008	1.003
14			1.0	135	1.000	1.031	1.139	1.000	1.092	1.051
15			5.0	60	1.000	1.136	1.228	1.000	1.092	1.098
16			10.0	35	1.000	1.122	1.243	1.000	1.109	1.127

in table 4.6 and 4.7, superiority of our proposed processor assignment is found. The number of PEs is limited by imposing the lower bound for each PE (cluster). Further, with taking heterogeneity of each PE into account, it is found that the lower bound for each PE should be adjusted to minimize the schedule length. The approach A is the combination of those two concepts. Thus, it is concluded that those two concepts are necessary for achieving effective use of processors.

4.7 Conclusion

In this chapter, we presented how to adjust each assignment unit size according to each PE to be assigned. At first, similar to the chapter 3, we defined the indicative value having effect on the schedule length. Then we proved the relationships between the indicative value and the schedule length, i.e., the reduction of the indicative value can lead to both the reduction of the lower bound and upper bound of the schedule length. Thus, the objective of our proposal in this chapter is to minimize the indicative value. Then we proposed the task clustering algorithm and the processor assignment algorithm to achieve effective use of processors.

Experimental comparisons by simulations show that both the indicative value and the schedule length can be reduced than other conventional approaches for heterogeneous distributed systems. Thus, it is concluded that experimental results and the theoretical results are matched to some extent.

The contribution of this chapter is to theoretically derive the lower bound for each assignment unit size for effective use of processors.

Chapter 5

Conclusion

In this dissertation, we proposed methods for achieving effective use of processors in both homogeneous distributed systems and heterogeneous distributed systems, where each processor is completely connected over the network. The fundamental issue in this dissertation is how to decide each assignment unit size for each processor. Our basic concept is to impose a certain condition to each assignment unit size to achieve effective use of processors. The constraint in this dissertation is to impose the lower bound for each assignment unit size. Thus, objective of our proposal is to minimize the response time under the condition that every assignment unit size exceeds the lower bound.

In chapter 3, homogeneous distributed systems is assumed. We defined the indicaive value, i.e., $sl_w(G^s_{cls})$ and then we proved that minimizing $sl_w(G^s_{cls})$ leads to the minimization of the response time. Thus, by deriving the lower bound when $sl_w(G^s_{cls})$ is minimized, the resultant schedule length can be small to some extent. Then we proposed the task clustering algorithm, in which every task is merged into one assignment unit until its size exceeds the lower bound. Experimental comparisons by simulations show that our proposal can provide better degree of effective use of processors than other conventional approaches.

In chapter 4, heterogeneous distributed systems is assumed. Similar to chapter 3, the indicative value, i.e., $sl_w(G^s_{cls}, \phi_s)$ is defined and then we proved that minimizing $sl_w(G^s_{cls}, \phi_s)$ can lead to the minimization of the schedule length. Since each task execution time and each data transfer time depend on each processing speed and each communication bandwidth, the lower bound should be set with considering the assigned processor's capability. Thus, our proposal in this chapter is to decide the lower bound on the path dominating $sl_w(G^s_{cls}, \phi_s)$ before the $(s+1)$-th task merging step. Since the lower bound is a function of a processing speed and a communication bandwidth, the actual lower bound can be obtained by assigning the lower bound into the upper bound of $sl_w(G^{s+1}_{cls}, \phi_{s+1})$ and specifying the processor by which the upper bound of $sl_w(G^{s+1}_{cls}, \phi_{s+1})$ can be minimized. After those two procedures have been completed, each task is merged into an assignment unit size (the sum of execution time for each task in the assignment unit size) until its size exceeds the lower bound. By repeating those procedures until every assignment unit size exceeds each lower bound, the task algorithm is completed. From experimental comparisons by simulations, it is found that the lower bound for each assignment unit size is more appropriate than other lower bounds decided by other policies. Moreover, similar to results in chapter 3, it is also found that $sl_w(G^{s+1}_{cls}, \phi_{s+1})$ by the proposal is smaller than other approaches, and so is true

for the response time.

From results obtained in chapter 3 and 4, it can be concluded that the new approach for effective use of processors in distributed systems is established by our proposal. The contribution by this dissertation is to propose a theoretical model to adjust each assignment unit size automatically.

The remained works are to expand the proposal to more realistic computing models, e.g., taking more dynamic communication conditions and the effect by hops into account. As for how to apply the proposal to realistic situation, the proposal must be implemented as programming libraries [46] or compiling infrastructures [47].

Acknowledgements

I am deeply grateful to my adviser, Prof. Yoshiyori Urano whose comments and suggestions were of inestimable value for my study. His continuous encouragement has been an invaluable support for me since I was a master course student.

I would like to express my hearty gratitude to Prof. Hidenori Nakazato whose comments made enormous contribution to my work. In the seminar held in every week, he has provided very fruitful academic and technical insights to me.

Prof. Wataru Kameyama and Prof. Young-Jin Park kindly provided valuable comments and advises to me. I do reflect their comments to not only my current work, but also my own future researches.

My best friend, Dr. Ye Kyaw Thu and me have shared many difficulties in doing researches since we were master course students. I would like to share new ideas with him and keep an humble attitude toward doing researches like him.

I would also like to express my gratitude to my parents for their moral support and warm encouragements.

Finally, I would like to express my gratitude to my wife, Mai, for her understandings of my situation and her devoted support.

Bibliography

[1] M. Girkar and C. D. Polychronopoulos, "Automatic Extranction of Functional Parallelism from Ordinary Programs," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 3 Issue 2, pp. 166 – 178, March 1992.

[2] M. Girkar and C. D. Polychronopoulos, "Extranting Task-Level Parallelism," *ACM Trans. on Programming Languages and Systems*, Vol. 17, No. 4, pp. 600 – 634, July 1995.

[3] C. D. Polychronopoulos, "The Hierarchical Task Graph and its Use in Auto-Scheduling," *Proc. of the 5th international conference on Supercomputing*, pp. 252 – 263, 1991.

[4] Y.K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, Vol.31, No.4, 1999.

[5] T. Tannenbaum, D. Wright, D. Miller, and M. Livny, "Condor - A Distributed Job Scheduler," *Beowulf Cluster Computing with Linux, The MIT Press*, 2002.

[6] I. Foster, "The Grid: A new infrastructure for 21st century science," *Physics Today*, Vol.55(2), pp. 42–47, 2002.

[7] I. Foster and C. Kesselman, "Computational Grids," *The Grid: Blueprint for a New Computing Infrastructure*, pp. 15 - 51, Morgan-Kaufman, 1999.

[8] "The Globus Alliance," available `athttp://www.globus.org/`.

[9] "UNICORE–Distributed computing and data resources," available at `http://www.unicore.eu/` .

[10] "SETI@home," available at `http://setiathome.berkeley.edu/` .

[11] "BOINC," available at `http://boinc.berkeley.edu/` .

[12] Y. K. Kwok and Ahmad I., "Static scheduling algorithms for allocating directed task graphs to multiprocessors ". *ACM Computing Surveys* , 31(4), 406-471, 1999.

[13] "Message Passing Interface Forum," available at `http://www.mpi-forum.org/`.

[14] "PVM: Parallel Virtual Machine," available at `http://www.csm.ornl.gov/pvm/`.

[15] "MPICH2," available at `http://www.mcs.anl.gov/research/projects/mpich2/`.

[16] O. Sinnen., "Task Scheduling for Parallel Systems," *Wiley*, 2007.

[17] A. Gerasoulis and T. Yang., "A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors," *Journal of Parallel and Distributed Computing*, Vol. 16, pp. 276-291, 1992.

[18] T. Yang and A. Gerasoulis., "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 9 pp. 951-967, 1994.

[19] H. Kanemitsu, G. Lee, H. Nakazato, T. Hoshiai, and Y. Urano, "Static Task Cluster Size Determination in Homogeneous Distributed Systems," *Proc. of The Fourth International Workshop on Automatic Performance Tuning (iWAPT2009)*, pp. 37-51, 2009.

[20] Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano, "Static Task Cluster Size Determination in Homogeneous Distributed Systems," *Software Automatic Tuning: from concepts to state-of-the-art results*, Springer-Verlag(ISBN: 1441969349), pp. 229 – 252, 2010.

[21] Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano, "On the Effect of Applying the Task Clustering for Identical Processor Utilization to Heterogeneous Systems," *Grid Computing*, InTech, ISBN:979-953-307-540-1 (to appear in March, 2012.).

[22] Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano, "A Task Clustering for Utilizing Computational Resources," *IPSJ Trans. on Advanced Computing Systems*, Vol. 4, No. 1, pp. 111-146, February 2011.

[23] Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano, "A Processor Mapping Strategy for Processor Utilization in a Heterogeneous Distributed System," *Journal of Computing*, Vol. 3, Issue 11, pp. 1 – 8, November 2011,

[24] A. Gerasoulis and T. Yang., "On the Granularity and Clustering of Directed Acyclic Task Graphs," *IEEE Trans. on Parallel And Distributed Systems*, Vol. 4, No. 6, June, 1993.

[25] T. Yang and A. Gerasoulis., "PYRROS: Static scheduling and code generation for message passing multiprocessors," *Proc. of the 6th ACM International Conference on Supercomputing*, pp. 428 – 437, 1992.

[26] J. C. Liou and M. A. Palis., "A Comparison of General Approaches to Multiprocessor Scheduling," *Proc. of the 11th International Symposium on Parallel Processing*, pp. 152 – 156, 1997.

[27] V. Sarkar., "Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors," Cambridge, *MA: MIT Press*, 1989.

[28] T. Yang and A. Gerasoulis., "List scheduling with and without communication delays," *Parallel Computing*, pp. 1321 – 1344, 1993.

[29] J. C. Liou, M. A. Palis., "An Efficient Task Clustering Heuristic for Scheduling DAGs on Multiprocessors," *Proc. of the 8th Symposium on Parallel and Distributed Processing*,October, 1996.

[30] M. Y. Wu and D. D. Gajski., "Hypertool: A programming aid for message-passing systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 1, No. 3, pp. 330– 343, 1990.

[31] Albert Y. Zomaya and Gerard Chan, "Efficient clustering for parallel tasks execution in distributed systems," *Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, pp. 167-175, 2004.

[32] Lepere. R and Trystram. D., "A New Clustering Algorithm for Large Communication Delays," *Proc. of the 16th Parallel and Distributed Processing Symposium(IPDPS2002)*, pp. 68-73, 2002.

[33] J. Pecero-Sanchez and D. Trystram., "A new Genetic Convex Clustering algorithm for parallel time minimization with large communication delays," *Proc. of the International Conference Parallel Computing (ParCo'2005)*, pp. 709-716, 2005.

[34] A. K. Amoura, E. Bampis and J-C. Konig., "Scheduling Algorithms for Parallel Gaussian Elimination With Communication Costs," *IEEE Trans. on Parallel and Distributed Systems*, vol. 9, no. 7, pp. 679-686, July 1998.

[35] T.H. Dunigan, "Performance of the Intel iPSC/860 and n-Cube 6400 Hypercubes," *Parallel Computing*, vol. 17, nos. 10 and 11, pp. 1285-1302, 1991.

[36] S. Chabridon and E. Gelenbe., "Dependable parallel computing with agents based on a task graph model," *Proc. of the 3rd Euromicro Workshop on Parallel and Distributed Processing*, pp.350 - 357, 1995.

[37] Y. Li, L. Zhao, et. al., "A Performance Model for Fast Fourier Transform," *Proc. of 2009 IEEE International Symposium on Parallel and Distributed Processing*, pp.1-11, 2009.

[38] M. I. Daoud and N. Kharma., "A high performance algorithm for static task scheduling in heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, Vol. 68, No. 4., pp. 399-409, 2008.

[39] H. Topcuoglu, S. H. Hariri and H. Y. Wu., "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 13, No. 3., pp. 260-274, 2002.

[40] N. Fujimoto, T. Baba, T. Hashimoto, and K. Hagihara, "On Message Packaging in Task Scheduling for Distributed Memory Parallel Machines," *International Journal of Foundations of Computer Science*, Vol. 12, No. 3, pp. 285 – 306, 2001.

[41] O. Sinnen and L. A. Sousa., "Communication Contention in Task Scheduling," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 16, No. 6., pp. 503-515, 2005.

[42] C. Boeres, J. V. Filho and V. E. F. Rebello, "A Cluster-based Strategy for Scheduling Task on Heterogeneous Processors," *Procs. of the 16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'04)*, pp.214 – 221, 2004.

[43] B. Cirou, E. Jeannot, "Triplet : a Clustering Scheduling Algorithm for Heterogeneous Systems," *Procs. of 2001 International Conference on Parallel Processing Workshops (ICPPW'01)*, pp. 231 – 236, 2001.

[44] S. Chingchit, M. Kumar and L.N. Bhuyan, "A Flexible Clustering and Scheduling Scheme for Efficient Parallel Computation," *Procs. of the 13th International and 10th Symposium on Parallel and Distributed Processing*, pp. 500 – 505, 1999.

[45] S. M. Bataineh, "Toward an analytical solution to task allocation, processor assignment, and performance evaluation of network processors," *Journal of Parallel and Distributed Computing*, Vol. 65, No. 1., pp. 29-47, 2005.

[46] "Parallel IT – True Object Oriented Parallelisation," available at `http://www.ece.auckland.ac.nz/~parallel/ParallelIT/`.

[47] "COINS Compiler Compiler Infrastructure," available at `http://www.coins-project.org/international/index.html`.

研　究　業　績

類 別	題名、　発表・発行掲載誌名、　発表・発行年月日、　連名者
○論文	A Processor Mapping Strategy for Processor Utilization in a Heterogeneous Distributed System, Journal of Computing, Vol. 3, Issue 11, pp. 1 – 8, November 2011, Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano.
論文	On the Effect of Applying the Task Clustering for Identical Processor Utilization to Heterogeneous Systems, Grid Computing, InTech, ISBN: 979-953-307-540-1, Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, andYoshiyori Urano (掲載決定).
○論文	計算資源の有効利用を目的としたタスククラスタリング, 情報処理学会論文誌 (ACS: コンピューティングシステム), pp. 111 – 146, Vol. 4, No. 1, 2011 年 2 月, 金光永煥, 李吉憲, 中里秀則, 星合隆成, 浦野義頼.
○論文	Static Task Cluster Size Determination in Homogeneous Distributed Systems, Software Automatic Tuning: from concepts to state-of-the-art results, Springer-Verlag (ISBN: 1441969349), pp. 229 – 252, 2010, Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano.
国際会議録	A Selection-based Item Display for Inquiring about the Long-Term Care Insurance System, Proc. of 10th International Conference on Intelligent Systems Design and Applications (ISDA 2010), pp. 795 – 800, 2010, Hidehiro Kanemitsu, Gao Hongyan and Yoshiyori Urano.
国際会議録	An User-Adaptive Symbol Presentation for Non-Verbal Communication, Proc. of 10th International Conference on Intelligent Systems Design and Applications (ISDA 2010), pp. 801 – 806, 2010, Gao Hongyan, Hidehiro Kanemitsu and Yoshiyori Urano.
国際会議録	Interactive Web Interface for Inquiring about Long-Term Care Insurance System, Proc. of 1st International Conference on Applied Bionics and Biomechanics (ICABB-2010) , October, 2010, Hidehiro Kanemitsu, Gao Hongyan and Yoshiyori Urano.
国際会議録	An User-Adaptive Symbol Presentation with AHP, Proc. of 1st International Conference on Applied Bionics and Biomechanics (ICABB-2010) ,October, 2010, Gao Hongyan, Hidehiro Kanemitsu and Yoshiyori Urano.
国際会議録	Static Task Cluster Size Determination in Homogeneous Distributed Systems, Proc. of The Fourth International Workshop on Automatic Performance Tuning (iWAPT2009), pp. 37-51, October, 2009, Hidehiro Kanemitsu, Gilhyon Lee, Hidenori Nakazato, Takashige Hoshiai, and Yoshiyori Urano.
国際会議録	A Monitoring Framework of WS-ResourceProperties in WS-GRAM, Proc. of IEEE International Conference on Web Services (IEEE ICWS 2008), pp. 629 —636, September, 2008, Hidehiro Kanemitsu, Yoshiyori Urano.
国際会議録	On the method for realizing globally and locally accessible resource Management for WS-GRAM, Proc. of IEEE International Conference on Web Services (IEEE ICWS 2007), pp.1201—1204, July, 2007, Hidehiro Kanemitsu, Yoshiyori Urano.

類別	題名、 発表・発行掲載誌名、 発表・発行年月日、 連名者
国際会議録	WSRF-based Global/Local Resource Mapping Architecture, Proc. of IEEE The 9th International Conference on Advanced Communication Technologies(ICACT2007), pp. 2061-2066, 2007, Hidehiro Kanemitsu, Yoshiyori Urano.
研究報告	地域活性化事業と地域の伝統産業とのコラボレーションにおける人的ネットワーク形成の効果に関する研究－桐生市の事例から－，情報社会学会誌，Vol.6 No.1，pp. 75 –87，2011 年，吉見 憲二，金光 永煥，豊川 正人，中里 秀則，星合 隆成，樋口 清秀.
研究報告	完全分散型 P2P-Grid におけるピアグルーピング手法の提案，信学技報，vol. 111，no. 117, CS2011-27, pp. 97-102, 2011 年 7 月，羅 維，金光永煥，王 歓，中里秀則.
研究報告	分散環境におけるプロセッサ数決定手法の性能評価，第 29 回電子情報通信学会アシュアランスシステム研究会技術研究報告，pp. 9 – 16，2010 年 3 月，金光永煥，李 吉憲，中里秀則，星合隆成，浦野義頼.
研究報告	Grid over P2P におけるダウンロード速度の向上のためのファイル転送手法の研究，信学技報, vol. 108, no. 457, NS2008-169, pp. 147-152, 2009 年 3 月，李 吉憲，大谷佳裕，金光永煥，中里秀則，富永英義.
研究報告	分散環境における最悪応答時間の改善を考慮したタスククラスタリング, 信学技報 ,vol. 108, no. 361, CPSY2008-45, pp. 13-18, 2008 年 12 月，金光永煥，盧 翊，大谷佳裕，李 吉憲，中里秀則，星合隆成，浦野義頼.
研究報告	P2P-Grid 環境におけるタスク割り当てのためのピアグループ構成手法，信学技報，vol. 108, no. 359, NS2008-111, pp. 19-22, 2008 年 12 月，盧 翊，金光永煥，大谷佳裕，李 吉憲，中里秀則，浦野義頼.
研究報告	実行粒度調整を目的とした分散処理タスクのクラスタリング，信学技報，vol. 108，no. 14, CPSY2008-2, pp. 7-12, 2008 年 4 月，金光 永煥，中里 秀則，星合隆成，浦野 義頼.
研究報告	タスク割り当て時の応答時間短縮を目的とした，静的なタスク実行粒度調整手法，信学技報, v ol. 107, no. 505, SS2007-70, pp. 79-84, 2008 年 3 月，金光 永煥，中里 秀則，星合隆成，浦野 義頼.
研究報告	汎用 P2P-grid における範囲限定型資源情報交換手法， 信学技報，vol. 107，no. 524, NS2007-162, pp. 179-182, 2008 年 3 月，岩崎 俊介，金光 永煥，中里 秀則， 星合 隆成， 富永 英義.
研究報告	P2P-grid における check point system の検討，信学技報 ，vol. 107，no. 524, NS2007-163, pp. 183-186, 2008 年 3 月，大谷佳裕，金光永煥，中里秀則，星合隆成，富永英義.
研究報告	地域情報化のための「情報還流型コミュニケーションモデル」の紹介，電子情報通信学会コミュニティ活性化時限専門研究委員会 第一回研究発表会予稿集， pp. 31—36，2008 年 2 月，星合隆成，金光永煥，中里秀則.

類別	題名、 発表・発行掲載誌名、 発表・発行年月日、 連名者
研究報告	P2P-gridにおける隣接ピア資源情報交換手法の検討, 信学技報 , vol. 107, no. 221,NS2007-53, pp. 1-4, 2007年 9月, 岩崎俊介, ガトパンデ アバイ, 金光永煥, 中里秀則, 星合隆成, 富永英義.
研究報告	Grid-Over-P2P環境における計算実行時間最適化のため のファイル転送手法の検討, 電子情報通信学会技術研究報告, vol.107, no.221, NS2007-54, pp.5-10, Sep. 2007, 近藤浩介, 中里秀則, ガトパンデ・アバイ, 金光永煥, 岩崎俊介, 星合隆成, 富永英義.
研究報告	WSRF/JMX によるGrid 環境の統一的な管理機能の実装,情報処理学会論文誌(プログラミング), Vol. 47/No. SIG1 1(PRO 30). p. 51, 2006年 7月, 金光 永煥, 浦野 義頼.
全国大会	本庄地域における介護推進に関する一考察,日本福祉のまちづくり学会第 12回全国大会概要集, pp. 412 – 413, 2009年 8月, 行木 雅子, 金光 永煥, 朱 槿, 浦野 義頼.
全国大会	P2P-grid における安定なチェックポイントシステムの提案 , 2009年電子情報通信学会総合大会講演論文集, B-19-20, p. 57, 2009年 3月, 大谷 佳裕, 李 吉憲, 金光 永煥, 中里 秀則, 富永 英義.
全国大会	介護保険制度の改定に対応する,介護情報の構造化手法: 2008年子情報通信学会大会講演論文集（情報・システム講演論文集 1）, p.43 , 2008年 3月, 金光 永煥, 行木 雅子, 浦野 義頼.
全国大会	P2P-gridにおける資源情報交換手法, 2008年電子情報通信学会大会講演論文集, p.111,2008年, 岩崎俊介, 金光永煥, 中里秀則, 星合隆成, 富永英義.
全国大会	BitTorrentを用いた依存ジョブ群の分散実行のためのファイル転送手法の検討, 2 008年電子情報通信学会大会講演論文集, p. 113, 2008年 , 近藤浩介, 金光永煥, 中里秀則, 星合隆成, 富永英義.
全国大会	地震災害情報の信頼度に対する、分散型評価手法, 日本災害情報学会第９回学会大会講演論文集、pp. 69 – 74, 2007年 11月, 金光 永煥, 行木 雅子, 門倉 博之, 浦野 義頼.
全国大会	地震災害時において周辺環境及び個人属性を考慮した避難経路の決定手法, 日本災害情報学会第 9回学会大会予稿集, pp. 75 – 80, 2007年 11月, 行木 雅子, 金光 永煥, 浦野 義頼
全国大会	Globus 上における統一的なリソース管理機能の考察, 第 4回情報科学技術フォーラム(FIT2005)講演論文集, pp. 167—169, 金光 永煥, 浦野 義頼.