JOURNAL OF SCHEDULING
J. Sched. 2002; **5**:379–394 (DOI: 10.1002/jos.116)

A standard task graph set for fair evaluation of multiprocessor scheduling algorithms

Takao Tobita[*,†] and Hironori Kasahara[‡]

Department of Electrical, Electronics and Computer Engineering, Waseda University, 3-4-1 Ohkubo, Shinjuku-ku, Tokyo 169-8555, Japan

SUMMARY

A ‘standard task graph set’ is proposed for fair evaluation of multiprocessor scheduling algorithms. Developers of multiprocessor scheduling algorithms usually evaluate them using randomly generated task graphs. This makes it difficult to compare the performance of algorithms developed in different research groups. To make it possible to evaluate algorithms under the same conditions so that their performances can be compared fairly, this paper proposes a standard task graph set covering many of the proposed task graph generation methods. This paper also evaluates as examples two heuristic algorithms (CP and CP/MISF), a practical sequential optimization algorithm (DF/IHS), and a practical parallel optimization algorithm (PDF/IHS) using the proposed standard task graph set. This set is available at http://www.kasahara.elec.waseda.ac.jp/schedule/. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: multiprocessor scheduling; task graph; performance evaluation; benchmark; optimization

1. INTRODUCTION

To efficiently execute programs in parallel on a multiprocessor system, a minimum-execution-time multiprocessor scheduling problem must be solved to determine the assignment of task sets to the processors and the execution order of the tasks so that the execution time is minimized [1].

For this ‘strong’ NP-hard scheduling problem, many scheduling algorithms have been proposed [1–20]. Hu’s algorithm [1, 2], a polynomial time optimization algorithm, works for simple problems where task processing times are all equal and the shape of the task graph is a tree. Coffman and Graham [1, 3] proposed a polynomial time optimization algorithm for another simple problem in which tasks with equal processing time are scheduled on two processors. Ramamoorthy *et al.* [4] proposed a search algorithm based on a dynamic programming (DP) algorithm that can generate optimal schedules for problems with about 40 tasks. Kasahara *et al.* [5] proposed a practical optimization algorithm, depth first/implicit heuristic search (DF/IHS), that can generate optimal schedules within 3 min on a HITAC M-280 main

*Correspondence to: Takao Tobita, Department of Electrical, Electronics and Computer Engineering, Waseda University, 3-4-1 Ohkubo, Shinjuku-ku, Tokyo 169-8555, Japan
†E-mail: tobita@oscar.elec.waseda.ac.jp
‡E-mail: kasahara@oscar.elec.waseda.ac.jp

frame computer system for about 75% of 300 large-scale task graphs with arbitrary shapes and up to 300 tasks having different task processing times.

The NP-hardness of the problem has led to the proposal of many heuristic algorithms such as critical path (CP) [1], critical path/most immediate successors first (CP/MISF) [5], and highest levels first with estimated times (HLFET) [6].

Bokhari [7], Chretienne [8], Palis *et al.* [9], and Varvarigou *et al.* [10] proposed polynomial time algorithms for tree-shaped task graphs considering interprocessor communication costs. Anger *et al.* [11] proposed the join latest predecessor (JLP) algorithm, a linear time optimization algorithm for an unlimited number of processors when communication costs are smaller than the shortest task processing time. Others include the earliest task first (ETF) heuristic algorithm proposed by Hwang *et al.* [12], the dominant sequence clustering (DSC) algorithm by Gerasoulis and Yang [13–17], the mapping heuristic (MH) algorithm by El-Rewini [18], the ready critical path (RCP) and RCP* algorithms by Yang *et al.* [19], the dynamic critical path (DCP) algorithm by Kwok *et al.* [20], and the task duplication based scheduling (TDS) algorithm by Darbha *et al.* [21].

These algorithms were evaluated using task graphs randomly generated or modelled from a few application programs. Because these task graphs are generally unavailable to other researchers, a fair comparison of the algorithms under the same conditions has been impossible. To eliminate this problem, this paper proposes a 'standard task graph set' that covers the proposed task-graph generation methods. The task graph set is used to evaluate the performance of two heuristic algorithms (CP and CP/MISF), a practical sequential optimization algorithm (DF/IHS), and its parallelized version (PDF/IHS).

2. TASK GRAPHS FOR MINIMUM EXECUTION TIME MULTIPROCESSOR SCHEDULING PROBLEM

The multiprocessor scheduling problem treated in this paper determines a non-preemptive schedule to minimize the execution time, or the schedule length, when a set of n computational tasks, $T=\{T_1,\dots,T_n\}$, having arbitrary precedence constraints and arbitrary processing times are assigned to m processors of the same capability. These tasks are represented by a directed acyclic graph (DAG) called a 'task graph', $G=(N,E)$, where N is the set of nodes ($|N|=n$ is the number of nodes) and E is the set of edges ($|E|=e$ is the number of edges), as shown in Figure 1. This problem has been known as strong NP-hard intractable optimization problem when it assumes arbitrary number of processors, arbitrary task processing time and arbitrary shapes of task graphs (as shown in Table I).

The performance of scheduling algorithms has been evaluated using randomly generated task graphs [4–6, 15, 17–19, 22–24] or task graphs modelled from a few application programs [6, 16, 17, 20–22, 25–27]. However, these task graphs are not typically available to other researchers. Therefore, fair performance comparisons of the algorithms under the same conditions have been impossible. To allow researchers to evaluate their algorithms under the same conditions, this paper proposes a standard task graph set covering previous task graph generation methods [4–6, 14, 17, 19, 23]. A prototype standard task graph set composed of 900 random task graphs, each with 50–2700 tasks, and several task graphs modelled from actual application programs are currently available at

http://www.kasahara.elec.waseda.ac.jp/schedule/.

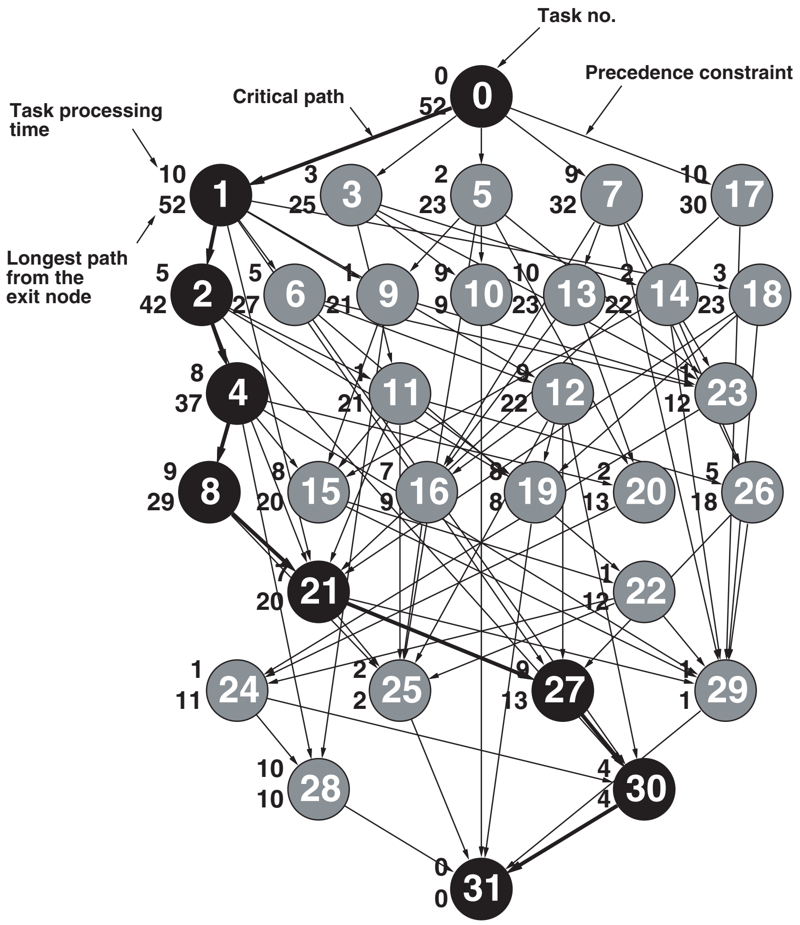

Figure 1. A task graph.

Table I. Complexity of scheduling problems.

Number of processors m	Task processing time T_i	Precedence constraints	Complexity
Arbitrary	Equal	Tree	$O(n)$ [2]
2	Equal	Arbitrary	$O(n^2)$ [3]
Arbitrary	Equal	Arbitrary	NP-hard
Fixed ($m \geqslant 2$)	$T_i = 1$ or 2 for all i	Arbitrary	NP-hard
Arbitrary	Arbitrary	Arbitrary	Strong NP-hard

The current prototype standard task graph set consists of task graphs without communication costs (i.e., 0 communication costs). Task graphs with communication costs are forthcoming.

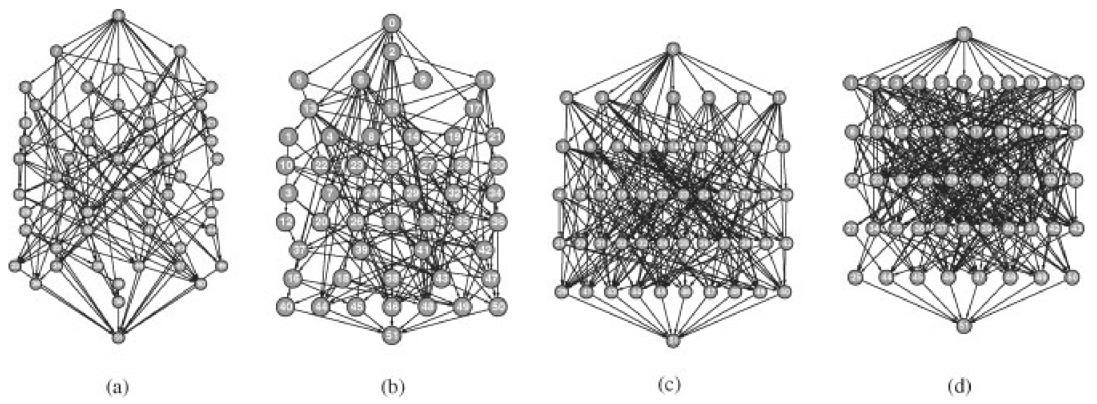

Figure 2. Examples of task graph shape: (a) sameprob; (b) samepred; (c) layrprob; and (d) layrpred.

2.1. *Task graphs generated randomly*

This section describes how the 900 random task graphs were generated in the prototype standard task graph set.

The graph size (in number of tasks) varies between 50 and 2700, such as 50, 100, 300, 500, 700, 900, 1100, 1300, 1500, 1700, 1900, 2100, 2300, 2500, and 2700.

Graph shapes (precedence constraints) were determined based on four different reported methods [6, 17, 23]. In the following, let A denote a task connection matrix with elements $a(i,j)$, where i, $0 \leqslant i \leqslant n+1$, and j, $0 \leqslant j \leqslant n+1$, represent the task number (T_0 is the entry dummy node and T_{n+1} is the exit dummy node). When $a(i,j)=1$, task T_i precedes task T_j. In other words, T_i must be completed before the execution of T_j is started (represented by '$T_i \prec T_j$'). On the other hand, when $a(i,j)=0$, T_i and T_j are independent of each other.

In the 'sameprob' edge connection method [23], $a(i,j)$ is determined by independent random values defined as follows:

$$\begin{aligned} P[a(i,j)=1] &= \pi \quad && \text{for } 1 \leqslant i < j \leqslant n \\ P[a(i,j)=0] &= 1-\pi \quad && \text{for } 1 \leqslant i < j \leqslant n \\ P[a(i,j)=0] &= 1 \quad && \text{if } i \geqslant j \end{aligned}$$

where π indicates the probability that there exists an edge (precedence constraint) between T_i and T_j. Figure 2(a) represents a random task graph having 50 tasks generated by the 'sameprob' method, where the edge connection probability (π) was specified as 0.1. In the current prototype standard task graph set, π was specified as 0.1 and 0.2.

However, with the 'sameprob' method, the number of predecessors increases with the number of tasks. Therefore, the standard task graph set also uses another predecessor-specification method, 'samepred'. The 'samepred' method specifies the average number of predecessors for each task. Figure 2(b) represents a random task graph having 50 tasks generated by the 'samepred' method, where the average number of predecessors was specified as 3. Currently, the average number of predecessors is set to 1, 3, or 5.

In another method, 'layrprob' [17], first the number of layers in the task graph is generated. Next, the number of independent tasks in each layer is randomly set. Finally, edges between layers are connected with the same probability π as with 'sameprob'. Figure 2(c) represents a random task graph having 50 tasks generated by the 'layrprob' method where the number of layers was specified as 5 and edge-connection probability π was specified as 0.2. Currently, the average width of each layer (the average number of tasks in each layer) is fixed at 10, and the number of layers is calculated by (number of tasks)/10.

The last method, 'layrpred', generates layers in the same way as with 'layrprob' and connects edges in the same way as with 'samepred'. Figure 2(d) represents a random task graph having 50 tasks generated by the 'layrpred' method, where the number of layers was specified as 5 and the average number of predecessors was specified as 5. Currently, the average width of each layer is fixed at 10, the number of layers is calculated by (number of tasks)/10, and the average number of predecessors is set to 1, 3, or 5.

The processing time of each task was determined using three methods. The mean value of the task processing time was fixed at 5 or 10 units of time (u.t.), and the processing time of each task was determined using uniformly distributed random numbers from 1 to 10 u.t. (mean task processing time of 5 u.t.) or from 1 to 20 u.t. (mean task processing time of 10 u.t.) [6]; using exponentially distributed random numbers [23]; or using normally distributed random numbers with a standard deviation of 1 (mean task processing time of 5 u.t.) or 3 (mean task processing time of 10 u.t.). These task processing time generation methods are identified using the names 'unifproc', 'expoproc', and 'normproc', respectively, in each task graph file.

We can thus select from 15 different task numbers, 10 edge-connection methods (precedence constraints form and average number of predecessors), 3 task processing time generation methods, and 2 average task processing times. The result is 900 possible task graphs.

2.2. Task graphs generated from actual application programs

The authors previously proposed a Fortran multigrain parallelization compilation method and have been developing the 'OSCAR FORTRAN Compiler' [28–30]. The standard task graph set includes several task graphs of actual application programs generated by this compiler. Also, previously reported task graphs modelled from actual application programs [25, 31] are available. The current standard task graph set includes task graphs for robot control (Figure 3), a sparse matrix solver (Figure 5), and a part of fpppp in the SPEC benchmarks (Figure 6).

2.2.1. Task graph for robot control. Figure 3 represents a task graph for Newton–Euler dynamic control calculation for the 6-degree-of-freedom Stanford manipulator [25, 31]. Figure 4 represents an optimal schedule (Gantt chart) of the task graph shown in Figure 3 for three processors, solved using the PDF/IHS algorithm described later.

2.2.2. Task graph for sparse matrix solver. Figure 5 represents a task graph for a random sparse matrix solver of an electronic circuit simulation that was generated using a symbolic generation technique and the OSCAR FORTRAN compiler with execution cost of a multi-processor system OSCAR.

2.2.3. Task graph for a part of SPEC fpppp program. Figure 6 represents a task graph for subroutine fpppp, which is part of the SPECfp95 benchmarks fpppp.

2.3. Standard task graph set format.

Figure 7 shows an example task graph file of the standard task graph set. It consists of a 'task graph part' and an 'information part'. The task graph part is the matrix of numbers at the top. The information part begins with '#' in the first column.

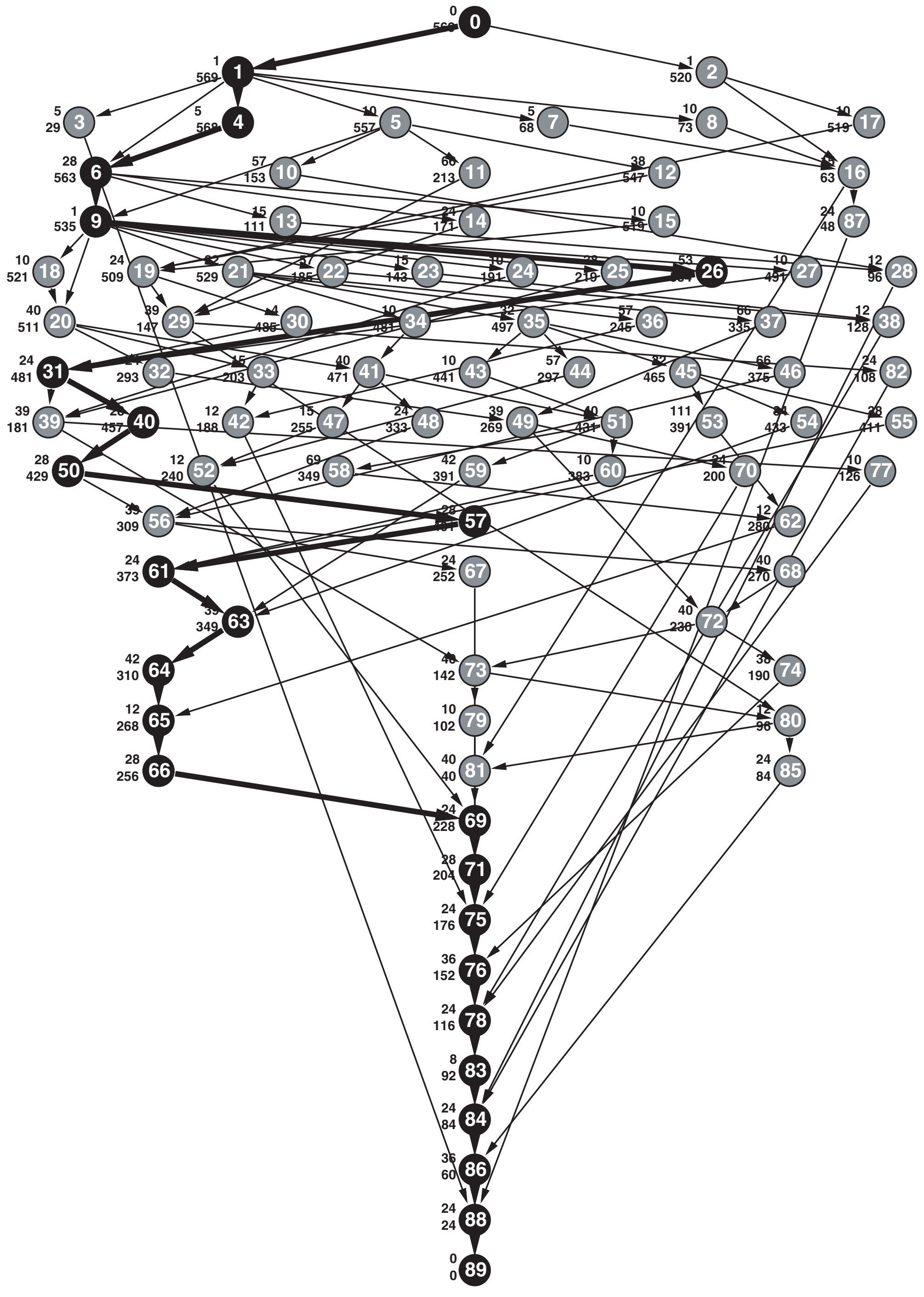

Figure 3. A task graph for Newton–Euler method for Stanford manipulator.

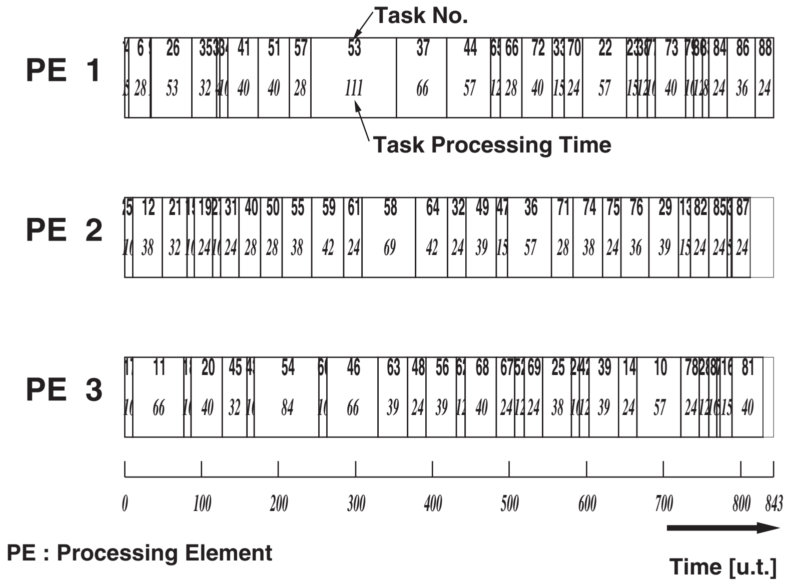

Figure 4. An optimal schedule of Figure 3 for 3 processors.

(1) *Task graph part*. The field for each number is 11 characters wide.
First line. The '10' represents the number of tasks.

Second line. This line holds the information of the entry dummy node (T_0). The first '0' represents task number 0. The second '0' means that the processing time for task 0 is zero. The third '0' means that task 0 has no predecessors.

Third line. The '1' represents task number 1. The '20' means that the processing time for task 1 is 20 u.t. The second '1' means that task 1 has one predecessor. The '0' means that the predecessor of task 1 is task 0.

Seventh line. The '5' represents task number 5. The '3' means that the processing time for task 5 is 3 u.t. The second '3' means that task 5 has three predecessors. The '2 3 4' mean that the predecessors of task 5 are tasks 2, 3, and 4.

Other lines. They represent the information for each task the same as above.

Last line. This line holds the information of the exit dummy node having 0 u.t. execution cost.

(2) *Information part*. The information part is composed of four different parts, namely, the common part (task graph file name etc.), precedence constraints form, task processing time, and other information such as critical path (CP) length and task graph parallelism.

(A) *Common information part*. The second line indicates that this task graph belongs to 'random' or 'application program' section and the name of the task graph. '10/sample.stg' means that the name of this task graph is 'sample.stg' in directory '10' of the standard task graph set archive.

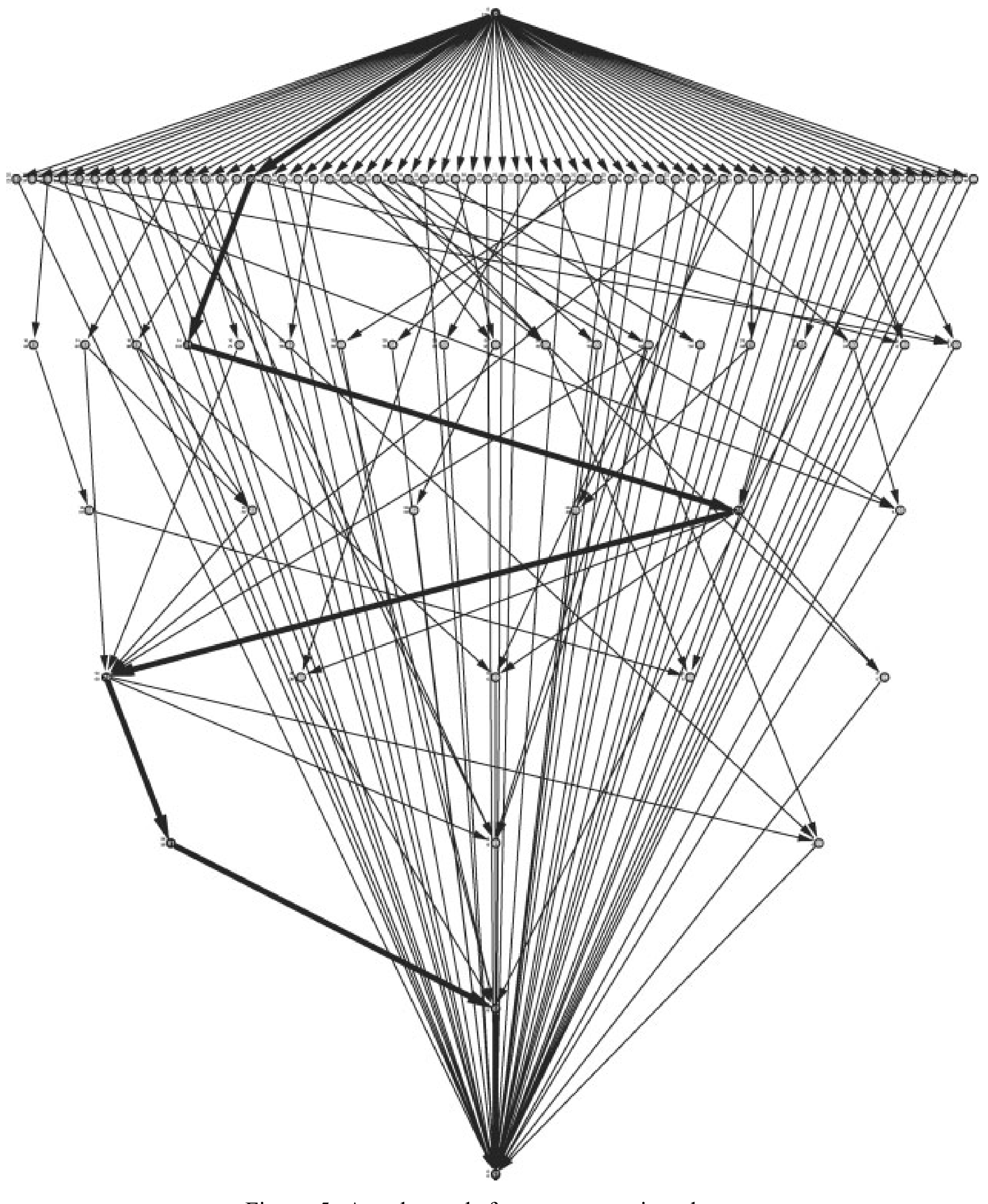

Figure 5. A task graph for sparse matrix solver.

(B) *Precedence constraints form information part*. This part explains the precedence constraints generation method, such as 'sameprob', 'samepred', 'layrprob', 'layrpred' and the name of the real application program. The following lines represent statistical information for the task graph, such as number of tasks, edges and predecessors, maximum and minimum

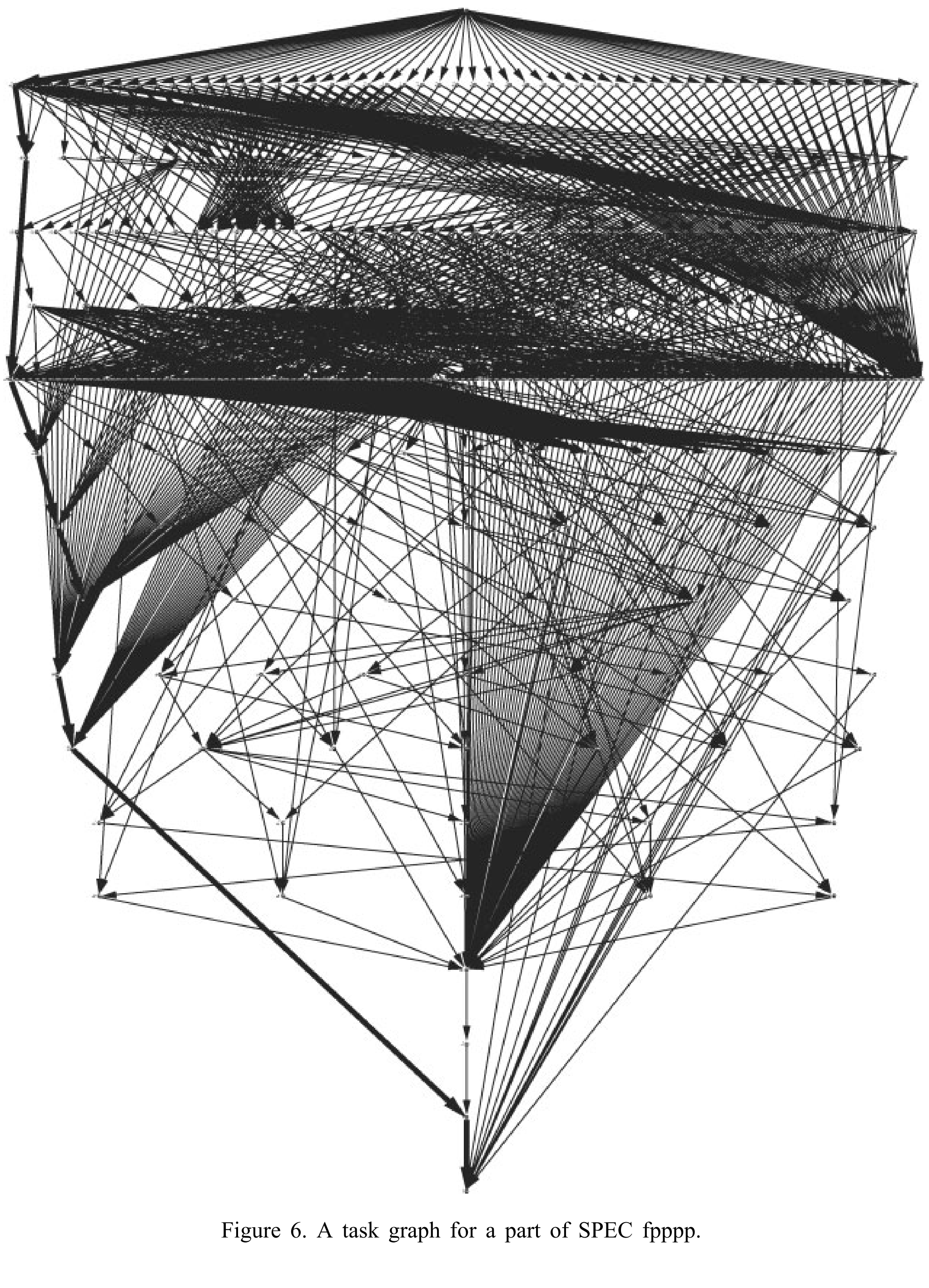

Figure 6. A task graph for a part of SPEC fpppp.

```
         10
          0         0         0
          1        20         1         0
          2         8         1         0
          3         8         1         0
          4         7         1         0
          5         3         3         2         3         4
          6        13         1         2
          7        13         2         1         4
          8        12         3         1         2         6
          9        15         2         1         5
         10        19         3         1         2         7
         11         0         3         8         9        10
# Standard Task Graph Set Project
# Random Task Graph 10/sample.stg
# Precedence constraints generator : layrprob
#   Random Seed       : 14343
#   Tasks             : 10 (+dummy tasks : 2)
#   Layers            : 3
#   Max. Layer Width  : 4
#   Min. Layer Width  : 3
#   Ave. Layer Width  : 3.333333
#   Edges             : 14 / 33 (+dummy edges : 7)
#   Max. Predecessors : 3
#   Min. Predecessors : 0
#   Ave. Predecessors : 1.400000
#   Probability       : 0.500000 (Real : 0.424242)
# Task processing time generator : unifproc
#   Random Seed       : 14344
#   Max. Proc. Time   : 20
#   Min. Proc. Time   : 3
#   Ave. Proc. Time   : 10.000000 (Real : 11.800000)
# CP Length           : 52
# Parallelism         : 2.269231
```

Figure 7. A task graph with 10 tasks.

number of predecessors, and so on. The ‘Probability’ line represents edge connection probability of all connectable edges. For example, in line 27 of Figure 7,

Probability: 0.500000 (Real: 0.424242)

indicates that the edge connection probability specified to the random task graph generator program was 0.500000; however, the real edge connection probability of the generated task graph is 0.424242.

(C) *Task processing time information part*. This part represents the task processing time generation method such as ‘unifproc’, ‘expoproc’, ‘normproc’ and the name of the real application program.

The following lines represent statistical information of task processing time, such as maximum, minimum or average values of task processing time. ‘Ave. Proc. Time’ line represents average amount of task processing time. At line 32 of Figure 7,

# Ave. Proc. Time: 10.000000 (Real: 11.800000)

indicates that the average task processing time specified for a random task processing time generator program was 10.000000; however, the real average task processing time of the generated task graph is 11.800000.

(D) *Other information part*. This part represents other information such as critical path length and parallelism. At line 33 of Figure 7,

# CP Length: 52

indicates that the critical path length (largest path length from the exit node to the entry node) is 52 u.t.

At line 34 of Figure 7,

# Parallelism: 2.269231

indicates that the parallelism of the task graph is 2.269231 which is calculated by (sum of all task processing time)/(critical path length).

3. EXAMPLES OF EVALUATION USING STANDARD TASK GRAPH SET

This section describes the performances of two heuristic algorithms such as CP and CP/MISF, a practical optimization algorithm such as DF/IHS (using 1 processor element), and PDF/IHS (using 3 or 6 processor elements) using the prototype standard task graph set.

3.1. Scheduling algorithms

The authors have proposed and evaluated a heuristic scheduling algorithm critical path/most immediate successors first (CP/MISF) [5], a practical sequential optimization algorithm depth first/implicit heuristic search (DF/IHS) [5] and a practical parallel optimization algorithm parallelized DF/IHS (PDF/IHS).

CP/MISF method is an improved version of the CP method [1], which gives the highest priority to a task with the largest path length from the exit node. CP/MISF gives the highest priority for a task that has the most immediate successors to break a tie if several tasks have the same path length in CP algorithm.

The DF/IHS algorithm is a depth first branch and bound algorithm, that takes advantage of the best heuristic algorithm CP/MISF and sharp lower bounds, such as Fernández’s expansion of Hu’s lower bound [5, 32]. Before the DF/IHS algorithm, for this non-preemptive general scheduling problem with arbitrary shape of precedence constraint, arbitrary task processing time and arbitrary number of processors, optimal solutions could be obtained only when the number of processors was restricted to 2 and the number of tasks was restricted to about 40 by using DP algorithm [4]. However, using DF/IHS, it was confirmed that optimal schedules were found for about 75% of the problems, approximate solutions having errors of less than 5% were found for about 92% of the problems, and approximate solutions having errors of less than 10% from optimal solutions were found for all problems having several hundred tasks on a HITAC M-280 system within 3 min. It is also known that DF/IHS is effective for scheduling problems of various objective functions such as the minimum total weighted flow time multiprocessor scheduling algorithm [33]. Furthermore, it has been confirmed that parallel processing of various applications like robot-arm control computations could be implemented efficiently on real multiprocessor systems using DF/IHS algorithm [25, 34, 35].

However, since this scheduling problem is a strong NP-hard problem as mentioned before, there exist problem instances that require an extremely large amount of processing time to find an optimal solution. As a method for quickly solving such problems, and thus decreasing the time required for scheduling and solving larger problems, the authors have developed PDF/IHS, which parallelizes the search part of DF/IHS. PDF/IHS parallelizes DF/IHS using a 'hierarchical pincers attack search' from both sides of the search tree, assigning a partial search tree to minimize communication overhead, and balancing loads among processors.

3.2. Evaluations of scheduling algorithms

This section describes the performance of heuristic algorithms CP and CP/MISF, a practical sequential optimization algorithm DF/IHS, and a parallel optimization algorithm PDF/IHS on 3 or 6 processor elements (PEs) using the proposed standard task graph set. In this paper, 660 task graphs having 50–1900 tasks in the standard task graph set are scheduled to 10 processors. The performance of DF/IHS and PDF/IHS is evaluated on a shared main memory multiprocessor system Sun Ultra Enterprise 3000 (E3000). E3000 has 6 Ultra SPARC 167 MHz CPUs (Processor Elements, or PEs) and 384 MB shared main memory. Figure 8 represents the architecture of the E3000.

The upper limit of search time is set to 600 s (10 min) because this scheduling problem is NP-hard and there exist problem instances which require incredibly large processing time to find an optimal solution.

In addition, the performances of heuristic algorithms are also evaluated on Sun Ultra 60 Workstation having Ultra SPARC-II processor (360 MHz) and 128 MB main memory.

Table II represents the number of cases in which optimal solutions are obtained from 60 total cases for each number of tasks. As shown in Table II, heuristic algorithms (CP and CP/MISF) obtain optimal solutions for about 75% of 660 tested task graphs, whereas DF/IHS obtains optimal solutions for 83.79% within 600 s, predetermined upper limit of search time. Furthermore, PDF/IHS obtains optimal solutions for 95.61 and 96.06% with 3 and 6 PEs, respectively. These results show the advantage of parallel search of PDF/IHS clearly. Also, DF/IHS and PDF/IHS can obtain optimal solutions in a short time for the task graphs with more than 1,000 tasks that no one has been able to solve before.

As for the difference of task graph shape, 24 cases of 26 problems wherein PDF/IHS with 6 PEs cannot get optimal solutions are made by 'layrprob' method. Thus, it appears that the problems scheduling task graphs made by 'layrprob' method are more intractable than other task graphs for DF/IHS and PDF/IHS.

Next, we investigate the impact of the difference of task graphs on the performance when the parameters for random task graph generation are changed. Table III represents the relationship between the optimal solutions rate and task execution time selection method. As shown in the table, task graphs having task execution time determined by normally distributed

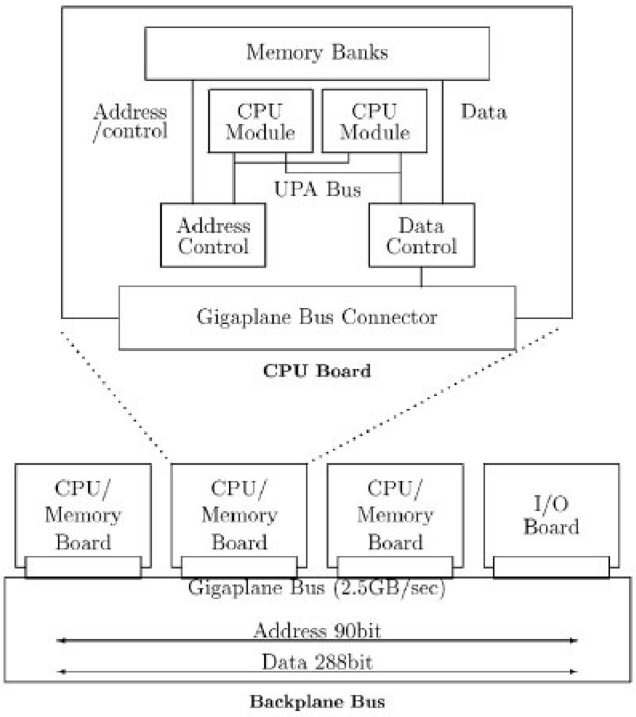

Figure 8. Architecture of ultra enterprise 3000.

Table II. Number of cases wherein optimal solutions are obtained for each number of tasks.

					PDF/IHS	
Tasks	Cases	CP	CP/MISF	DF/IHS	3PEs	6PEs
50	60	54	54	57	60	60
100	60	47	49	53	60	60
300	60	38	38	50	58	59
500	60	44	42	48	56	57
700	60	44	45	52	58	58
900	60	47	46	48	56	57
1100	60	44	44	48	57	57
1300	60	43	44	50	58	58
1500	60	44	45	47	55	55
1700	60	43	44	46	55	55
1900	60	48	48	54	58	58
Total	660	496	499	553	631	634
%	100.00	75.15	75.61	83.79	95.61	96.06

random numbers are difficult since normally distributed random numbers tend to become close to average task processing time compared with uniformly or exponentially distributed numbers.

Table III. Optimal schedule rate (by task processing time).

Distribution	Cases	CP	CP/MISF	DF/IHS	PDF/IHS	
					3PEs	6PEs
Uniform	220	200	202	215	217	217
Exponential	220	215	214	220	220	220
Normal	220	81	83	118	194	197
Total	660	496	499	553	631	634
%	100.00	75.15	75.61	83.79	95.61	96.06

Table IV. Parallelism of task graphs and margin from lower bounds.

Algorithm	Parallelism Para		Difference			
			u.t.		%	
CP	5.98	228.91	1	22	0.05	9.62
CP/MISF	5.98	228.91	1	21	0.05	7.69
DF/IHS	8.15	228.91	1	19	0.05	5.77
PDF/IHS (3PEs)	8.15	68.59	1	19	0.09	2.20
PDF/IHS (6PEs)	8.15	68.59	1	19	0.09	2.20
All Problems	2.26	237.63				

Next, the relationship between optimal rate and task graphs' parallelism (*Para*) is examined. Table IV represents the parallelism and the difference between lower bound and provisional solution that was obtained after 10 min search. In all the 660 tested cases, Para is distributed between 2.26 and 237.63 as shown in Table IV. The parameter Para of the problems that could not find the optimal solutions ranges between 5.98 and 228.91 for CP and CP/MISF algorithms, between 8.15 and 228.91 for DF/IHS, and between 8.15 and 68.59 for PDF/IHS with 3 or 6 PEs, respectively. Additionally, Para of the problems having larger than 10 u.t. difference are in the range from 8.40 to 10.44. It means that it is difficult to find optimal solutions when the number of processors is close to the parallelism of task graphs.

Table IV also shows the percentage of difference between the provisional solution, or the best solution found in 10 min, and the lower bound. This result means that PDF/IHS can obtain optimal or approximate solutions having errors of less than 3% from optimal solutions for all tested cases within 10 min.

Finally, Table V represents the scheduled results when an actual application task graph of subroutine fpppp in SPECfp95 benchmarks fpppp is assigned to 2 processors. As shown in Table V, search times of the heuristic algorithms are 0.013 s and 0.009 s for CP and CP/MISF, respectively, and the schedule lengths are 3575 and 3576 u.t., respectively, which are shorter than that of FIFO, 3606 u.t. Moreover, although DF/IHS and PDF/IHS with 3 PEs do not obtain an optimal solution within 10 min predetermined upper limit, PDF/IHS with 6 PEs could obtain an optimal solution of 3565 u.t. in 2.684 s search time.

Table V. Scheduled results for fpppp.

	FIFO	CP	CP/MISF	DF/IHS	PDF/IHS	
					3PEs	6PEs
Schedule length	3606	3575	3576	(3576)	(3575)	3565
Search time (s)	0.0027	0.0125	0.0090	600.7214	600.9855	2.6837

( ) denotes the provisional solution within 10 min.

4. CONCLUSIONS

This paper has proposed a standard task graph set for fair evaluation of multiprocessor scheduling algorithms taking into consideration the previous literature for multiprocessor scheduling algorithms and their evaluation. Currently, the prototype standard task graph set is available at the following Website:

http://www.kasahara.elec.waseda.ac.jp/schedule/.

Also, as an example of performance evaluation, this paper has evaluated the scheduling algorithms, such as heuristic algorithms CP and CP/MISF, and practical optimization algorithms DF/IHS and PDF/IHS, using the 'standard task graph set'.

As a result of the evaluation, PDF/IHS using 6 PEs could obtain an optimal solution for more than 96% of the 660 tested task graphs. Also, heuristic algorithms could obtain optimal solutions for about 75% of the problems. However, it was confirmed that the problems generated by 'layrprob', having normally distributed task execution time or its parallelism being close to the number of processors, it is very difficult to obtain optimal solutions for DF/IHS and PDF/IHS.

The authors are currently improving the 'standard task graph set' to include other task graph generation methods, task graphs modelled real application programs, and task graphs considering communication costs. The performance evaluation of other algorithms using these task graphs is also a future work.

To make the standard task graph set better, the authors welcome information or URL's about other task graphs. Please send information to

STG@oscar.elec.waseda.ac.jp.

REFERENCES

1. Coffman EG. *Computer and Job-shop Scheduling Theory*. Wiley: New York, 1976.
2. Hu TC. Parallel sequencing and assembly line problem. *Operations Research* 1961; **9**:841–848.
3. Coffman EG, Graham RL. Optimal scheduling for two-processor systems. *Acta Informatica* 1972; **1**(3): 200–213.
4. Ramamoorthy CV, Chandy KM, Gonzalez MJ. Optimal scheduling strategies in a multiprocessor system. *IEEE Transactions on Computers* 1972; **C-21**(2):137–146.
5. Kasahara H, Narita S. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Transactions on Computers* 1984; **C-33**(11):1023–1029.
6. Adam TL, Chandy KM, Dickson JR. A comparison of list schedules for parallel processing systems. *Communications of the ACM* 1974; **17**(12):685–690.
7. Bokhari SH. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Transactions on Software Engineering* 1981; **SE-7**(6):583–589.

8. Chretienne P. A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *European Journal of Operational Research* 1989; **43**:225–230.
9. Palis MA, Liou J, Wei DSL. Task clustering and scheduling for distributed memory parallel architectures. *IEEE Transactions on Parallel and Distributed Systems* 1998; **7**(1):46–55.
10. Varvarigou TA, Roychowdhury VP, Kailath T, Lawler E. Scheduling in and out forests in the presence of communication delays. *IEEE Transactions on Parallel and Distributed Systems* 1996; **7**(10):1065–1074.
11. Anger FD, Hwang J, Chow Y. Scheduling with sufficient loosely coupled processors. *Journal of Parallel and Distributed Computing* 1990; **9**:87–92.
12. Hwang J, Chow Y, Anger FD, Lee C. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing* 1989; **18**(2):244–257.
13. Gerasoulis A, Venugopal S, Yang T. Clustering task graphs for message passing architectures. In *Proceedings of the 4th ACM International Conference of Supercomputing* 1990; 447–456.
14. Yang T, Gerasoulis A. A fast static scheduling algorithm for dags on an unbounded number of processors. In *Proceedings of Supercomputing '91*, 1991; 633–642.
15. Gerasoulis A, Yang T. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors. *Journal of Parallel and Distributed Computing* 1992; **16**:276–291.
16. Gerasoulis A, Yang T. On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and Distributed Systems* 1993; **4**(6):686–701.
17. Yang T, Gerasoulis A. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Transactions on Parallel and Distributed Systems* 1994; **5**(9):951–967.
18. El-Rewini H, Lewis TG. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing* 1990; **9**:138–153.
19. Yang T, Gerasoulis A. List scheduling with and without communication delays. *Parallel Computing* 1993; **19**(12):1321–1344.
20. Kwok Y, Ahmad I. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 1996; **7**(5):506–521.
21. Darbha S, Agrawal DP. Optimal scheduling algorithm for distributed-memory machines. *IEEE Transactions on Parallel and Distributed Systems* 1998; **9**(1):87–95.
22. Shirazi B, Wang M, Pathak G. Analysis and evaluation of heuristic methods for static task scheduling. *Journal of Parallel and Distributed Computing* 1990; **10**:222–232.
23. Almeida VAF, Vasconcelos IMM, Árabe JNC, Menascé DA. Using random task graphs to investigate the potential benefits of heterogeneity in parallel systems. In *Proceedings of Supercomputing '92* 1992; 683–691.
24. Manimaran G, Siva Ram Murthy C. An efficient dynamic scheduling algorithm for multiprocessor real-time systems. *IEEE Transactions on Parallel and Distributed Systems* 1998; **9**(3):312–319.
25. Kasahara H, Narita S. Parallel processing of robot-arm control computation on a multimicroprocessor system. *IEEE Journal of Robotics and Automation* 1985; **RA-1**(2):104–113.
26. Llosa J, Valero M, Ayguadé E, González A. Modulo scheduling with reduced register pressure. *IEEE Transactions on Computers* 1998; **47**(6):625–638.
27. Hurson AR, Lee B, Shirazi B, Wang M. A program allocation scheme for data flow computers. In *Proceedings of the 1990 International Conference on Parallel Processing*, 1990; I 415–I 423.
28. Yoshida A, Koshizuka K, Kasahara H. Data-localization for fortran macrodataflow computation using partial static task assignment. In *Proceedings of the 10th ACM International Conference on Supercomputing* 1996; 61–68.
29. Kasahara H, Honda H, Narita S. Parallel processing of near fine grain tasks using static scheduling on oscar. In *Proceedings of the IEEE ACM Supercomputing '90*, November 1990.
30. Kasahra H, Honda H, Mogi A, Ogura A, Fujiwara K, Narita S. A multi-grain parallelizing compilation scheme for oscar (optimally scheduled advanced multiprocessor). In *Proceedings of the 4th Workshop on Languages and Compilers for Parallel Computing*, August 1991.
31. Kasahara H. Parallel processing of robot arm dynamic control computation on multimicroprocessors. *Microprocessors and Microsystems* 1990; **14**(1):3–9.
32. Fernández EB, Bussell B. Bounds on the number of processors and time for multiprocessor optimal schedules. *IEEE Transactions on Computers* 1973; **C-22**(8):745–751.
33. Kasahara H, Wada H, Kai M, Narita S. An application of df/ihs to minimum total weighted flow time multiprocessor scheduling problem. *Transactions of the Institute of Electronics, Information and Communication Engineers, Japan* 1987; **J70-D**(6):1083–1091.
34. Kasahara H, Fujii T, Nakayama H, Narita S, Chua LO. A parallel processing scheme for the solution of sparse linear equations using static optimal-multiprocessor-scheduling algorithms. In *Proceedings of the 2nd International Conference on Supercomputing*, May 1987.
35. Kasahara H, Narita S. An approach to supercomputing using multiprocessor scheduling algorithms. In *Proceedings of the IEEE 1st International Conference on Supercomputing*, December 1985; 139–148.