The Eighth International Conference on Intelligent Technologies (InTech'07)
Sydney, Australia, 12-14 December 2007

A Hybrid Evolutionary Approach for Heterogeneous Multiprocessor scheduling

K. C. Tan, C. K. Goh, E. J. Teoh and D. K. Liu

***Abstract*— This paper considers the assignment of tasks with interdependencies in a heterogeneous multiprocessor environment where task execution time varies with task as well as the processing element processing it. The solution to this heterogeneous multiprocessor scheduling problem involves the optimization of complete task assignments and processing order within the assigned processors with minimum makespan, subject to the precedence constraint. To solve such a NP-hard combinatorial optimization problem, this paper presents a hybrid evolutionary algorithm that incorporates two local search heuristics that exploits the intrinsic structure of the solution as well as specialized genetic operators to encourage exploration of the search space. The effectiveness and contribution of the proposed features are validated on a set of benchmark problems characterized by different degrees of communication times, task and processor heterogeneities. Simulation results demonstrate the algorithm is capable of finding useful schedules on the set of new benchmark problems.**

***Index Terms*— Multiprocessor scheduling, heterogeneous, hybrid evolutionary algorithm, local search, precedence**

I. Introduction

THE emergence of computer programs with increasingly higher computational requirements and algorithmic complexity has necessitated the need for parallel processing elements in a multi-computer environment, which in turn has seen the increasing need for task allocation to be optimally distributed in a suitable manner to these individual processing units. The multiprocessor scheduling problem is a class of NP-hard combinatorial optimization problems [13], [17], [21], [26] and it can be categorized into different classes based on the characteristics of the program, the tasks to be scheduled, the multiprocessor system, as well as the availability of information [19], [20], [9]. Typically the processing elements constituting the multi-computer environment can be of the same capability or of a different capability. This paper is focused on the latter. The problem becomes even more challenging when communication delays are accounted for.

In a nutshell, the goal of a scheduler is to assign partitioned tasks to available processors in such a manner that not only are the requirements of precedence between these tasks are met, but also in addition to the objective in obtaining as minimal as possible a makespan [29]. Presently, there are numerous methods and approaches which have been developed and subsequently applied to the multiprocessor scheduling problem, typically using a deterministic approach. El-Rewini *et al* [9] provides a fairly comprehensive taxonomy of how scheduling problems can be categorized, and highlights the key differences that distinguishes one class from the next. Further to this, in [19], [20], Kwok and Ahmad present a wide-ranging overview and classification of scheduling algorithms, particularly focusing on deterministic and static scheduling problems. Most of the present techniques are based on heuristics [18], [22] that are not only greedy in nature but also capable of solving certain instances of the scheduling problem efficiently.

Evolutionary algorithm (EA) is a class of stochastic global optimization technique has been applied to solve the heterogenous multi-processor scheduling (HMPS) optimization problem. EAs are excellent global search algorithms but they can take a relatively long time to locate the local optimum in the region of convergence [25]. On the other hand, local search heuristics are capable of locating the optimum quickly but are prone to local optimal traps. Therefore, researcher often hybridized EAs with local search heuristics to maintain a balance between exploration and exploitation to improve the optimization processes [6], [12], [16], [23], [24].

This paper presents a new hybrid evolutionary algorithm (HEA) for the HMPS optimization problem. The proposed algorithm incorporates two local search operators, based on list scheduling and task duplication, to exploit the intrinsic structure of the scheduling problem. Unlike existing evolutionary approaches to the HMPS problem, the proposed HEA also implements a variable length chromosome which preserves the precedence relations, a PE schedule crossover which facilitates the exchange of good schedules assigned to the individual processors as well as specialized mutation operators to improve the diversity of the evolving population.

This paper is organized as follows: Section II gives an overview of existing works as well as the problem formulation of the HMPS. Section III presents the various features of the proposed HEA including the local search heuristics and specialized genetic operators as well as the algorithmic flow. Section IV presents the extensive simulation results and analysis of the proposed algorithm. Conclusions are drawn in Section V.

II. Heterogenous Multi-processor Scheduling Problem

A. Problem Formulation

The multiprocessor scheduling problem can be simply stated as follows:

> Assuming there are n tasks that have to be executed on m processors - where and when should each task

K. C. Tan, C. K. Goh and E. J. Teoh are with the Department of Electrical and Computer Engineering, National University of Singapore, 4, Engineering Drive 3, Singapore 117576 (e-mail: {ckgoh,eletankc}@nus.edu.sg). D. K. Liu is with the ARC Centre of Excellence for Autonomous Systems (CAS), Faculty of Engineering, University of Technology, Sydney, PO Box 123, Broadway, NSW 2007, Australia

be executed, such that some performance measure(s) is (are) optimized?

The task of the scheduling algorithm is to ultimately minimize a given cost function of time. The objective function used in this paper is defined as:

$$F(\mathbf{x}) = \arg\min\{\arg\max_{p} \zeta_f(p)\} \quad (1)$$

where $\zeta_f(p)$ denotes the time for processor p to finish the last task. The starting time $\zeta_s(t_i)$, of each task, t_i, at processor p_k is determined by the finish time of its predecessors and the time required to transfer the required data. This starting time is given by the following:

$$\zeta_s(t_i) = \arg\max_{ti} \zeta_f(t_j) + \zeta_c(t_i, t_j) \quad (2)$$

where $\zeta_c(t_i, t_j)$ is calculated as

$$\zeta_c(t_i, t_j) = c_{ij} d_{ij} \quad (3)$$

The goal of task assignment is to determine an assignment of tasks to processors and an order in which tasks are executed to optimize some performance measures. Often, the assignment process should aim to minimize the makespan. An optimal assignment determines both the allocation and the schedule (execution order) of each task.

A task in turn, is a collection of instructions, procedures or subroutines, possibly together with some data. Each task is assumed to be immutable. While distributing the tasks to parallel PEs is not difficult, introducing dependencies between the tasks causes degradation of the overall system performance. There are dependencies between some pairs of tasks since a procedure in one task may need to transfer control to another procedure in a different task or access data contained/produced in a different task. Further these tasks incur an communication delay when they are assigned to different PEs.

This paper considers deterministic scheduling, i.e. the duration of each task is known as well as precedence relations among tasks. In addition, if dependent tasks are executed on different processors, communication delays that are given in advance are also considered. These latencies also include memory access and synchronization delays. In addition, we also consider a heterogeneous system, that is a multiprocessor environment consisting of processors with different capabilities. Moreover, we only consider a non-preemptive system, that is, each PE will complete the processing of each task that is assigned to it. Essentially, this means that PEs will not suspend its processing to take on another task.

B. Problem Generator

In order to verify the efficacy of our proposed approach, a benchmark generator is constructed to generate a set of test problems for the experimental study. In constructing these problems artificially, and in a random manner, the input variables essentially controls not only the size, but also the complexity of the generated problem set. Specifically, these variables are:

1) the number of nodes/tasks,
2) the number of processors available,
3) the degree of network connectivity,
4) the communication-to-computation ratio,
5) the mean processing time,
6) the variance of processing time,
7) the degree of heterogeneity,
8) the degree of precedence relationship.

Having said that, the generator produces different problem sets for a given set of input parameters. For similar set of parameter, different task problems are generated due to randomness. The input parameters to the generator are shown in Table I, together with the associated range of values. Using these inputs for the benchmark problem generator, sets of random DAGs were constructed to be used as the test bed problems in our experimental study. For our simulation study, 10 problem sets were generated using various combination of the above input parameters, and are listed in Table II.

While the standard multiprocessor scheduling problem is itself an NP-hard problem, additional factors such as communication delays and heterogeneity increases the complexity of the problem. Hence, due to the sheer number of potential solutions in the search space, scheduling becomes a complex task without the use of an effective search algorithm. These sets are classified in terms of the possible difficulties. Each test sets consists of different test problems with different degrees of heterogeneity and dependencies. Here, we consider a total of 10 test sets generated in this study, which differs in terms of degree of heterogeneity, density and CCR. A higher CCR value penalizes dependencies which require transmission or passing of messages from one processor to the next, making it less optimal for inter-processor communication to occur. The variance of processing time and degree of heterogeneity affects the individual processing capabilities of each processor, thus making 'slower' processors less likely to be assigned tasks, and biasing the utility of 'faster' processors. Lastly, the degree of dependency affects the total latency of the makespan in that each processor would have to 'wait' for its dependent tasks to finish execution.

III. Hybrid Evolutionary Algorithm

This section presents the hybrid evolutionary algorithm (HEA) specifically designed to solve the HMPSP by means of specialized genetic and local search operators. The procedure for generating the initial population is presented in Section III-A while Section III-B describes the structure of the variable-length chromosome used to encode the task schedule in the HEA. Sections III-C and III-D describe the specialized crossover and mutation operators used to explore the search space respectively. Two local search heuristics that exploit the intrinsic structures of a HMPSP solution are presented in Section III-E. Finally, the algorithmic flow of the HEA is presented in Section III-F.

A. Initialization

The initial population is built using a random list scheduling heuristic which ensures that the precedence relationships among the tasks are preserved. The initialization process starts with the assignment of priority to each task to be scheduled.

TABLE I

DESCRIPTION OF INPUTS TO TASK GENERATOR

Parameter	Description	Values
CCR	Communication-to-computation ratio	{0.5,1,1.5,2}
mean_{proc}	Mean processing time	{10}
h_{pe}	Variance of processing time	{0.25,0.5,0.75}
h_t	Degree of heterogeneity	{0.25,0.5,0.75}
dpe	Width of DAG	{0.5}
dt	Degree of dependency	{0.25,0.5,0.75}
n	Number of processors	{15}
m	Number of tasks	{100}

TABLE II

GENERATED PROBLEM SETS

Problem Set	CCR	mean_proc	hpe	ht	dpe	dt	n	m
1	0.5	10	0.25	0.25	0.5	0.5	15	100
2	1	10	0.25	0.25	0.5	0.5	15	100
3	1.5	10	0.25	0.25	0.5	0.5	15	100
4	2	10	0.25	0.25	0.5	0.5	15	100
5	1	10	0.25	0.25	0.5	0.25	15	100
6	1	10	0.25	0.25	0.5	0.75	15	100
7	1	10	0.5	0.25	0.5	0.5	15	100
8	1	10	0.75	0.25	0.5	0.5	15	100
9	1	10	0.25	0.5	0.5	0.5	15	100
10	1	10	0.25	0.75	0.5	0.5	15	100

In this paper, the priority of the i-th task is simply the sum of the number of its parent tasks and its' priority as given below

$$\text{Pr}_{T_i} = |\vec{P_i}| + \sum_{j \in |\vec{P_i}|} \text{Pr}_{T_j} \tag{4}$$

where $\vec{P_i}$ is the set of parent tasks of the i-th task.

The list of task is then sorted in the order of increasing priority. This prority list is also used during the genetic processes to maintain the precedence requirements. Instead of assigning the tasks to the earliest available PE, the lowest priority task is assigned to the PEs randomly. The rationale is to provide the initial population with a wider range of diversity to start with.

B. Variable PE Chromosome

EAs operates on a set of encoded parameters to explore the solution space, providing researchers with the flexibility to design an appropriate representation that fulfills some criteria such as ease of implementation or exploitation of the problem structure. For simplicity, the chromosome is often represented as a fixed-structure and the embedded variables are usually assumed to be independent and context insensitive. As mentioned before, the precedence relations among the tasks must be satisfied in the HMPSP. In [5], [27], the chromosome is a n-dimensional array denoting the n tasks to be allocated and the encoded variable in each element represents the PE scheduled to execute the associated task. While such an encoding scheme is simple to implement, it does not consider the order in which the various tasks are processed and the evolved schedules will not satisfy the precedence constraints. On the other hand, Wu *et al* [29] considered a representation which encodes task-processor pairs and the order in which the pairs appear in the chromosome determines the order in which the tasks will be performed on each processor.

In this paper, a variable length chromosome which encodes the complete schedule including the task allocation and the order of execution is implemented. Similar to [15], [30], the proposed representation encodes several computational task schedules, where each schedule represents the list and order of the task to be executed on the associated PE. Each computational task schedule will henceforth be denoted as PE schedule. In contrast to the mentioned works, the proposed variable length chromosome does not enforce a fixed number of schedules, i.e. the length of the chromosome varies with the actual number of PE utilized as shown in Fig. 1. As noted by Hou *et al* [15], this form of representation has two distinct advantages: 1) it maintains the precedence relations for the tasks executed in a PE and 2) considerations of precedence are confined to each PE schedule. Additionally, such a representation is efficient and facilitates the design of problem-specific genetic operators.

C. PE Schedule Crossover

The crossover operation applied by most EAs to solve HMPSP generally involve the swapping of random segments of tasks or processes between chromosomes which do not preserve the quality of the different PE schedules. It should be noted that the makespan of the multiprocessor schedule is dependent on the fitness of the constituent PE schedules. Therefore, in contrast to existing works, this paper adopts a crossover which allows good PE schedules to be shared with other chromosomes in the evolving population.

The operation of the crossover is illustrated in Fig. 2. In the PE schedule crossover, a random PE schedule from each parent is selected for crossover. In the case where one of the selected chromosomes has only one PE schedule, only a schedule associated with a different PE is selected and inserted from the other parent. The selected PE schedule of one parent will either be inserted into the other chromosome as a new

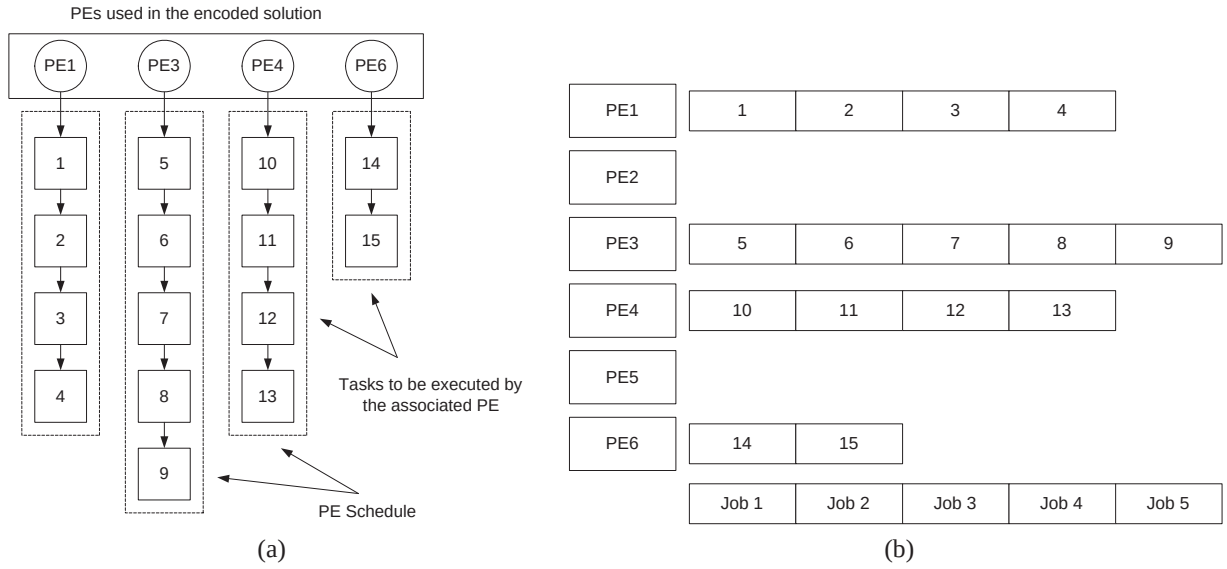

Fig. 1. Illustration of (a) the variable length chromosome and (b) the associated schedule.

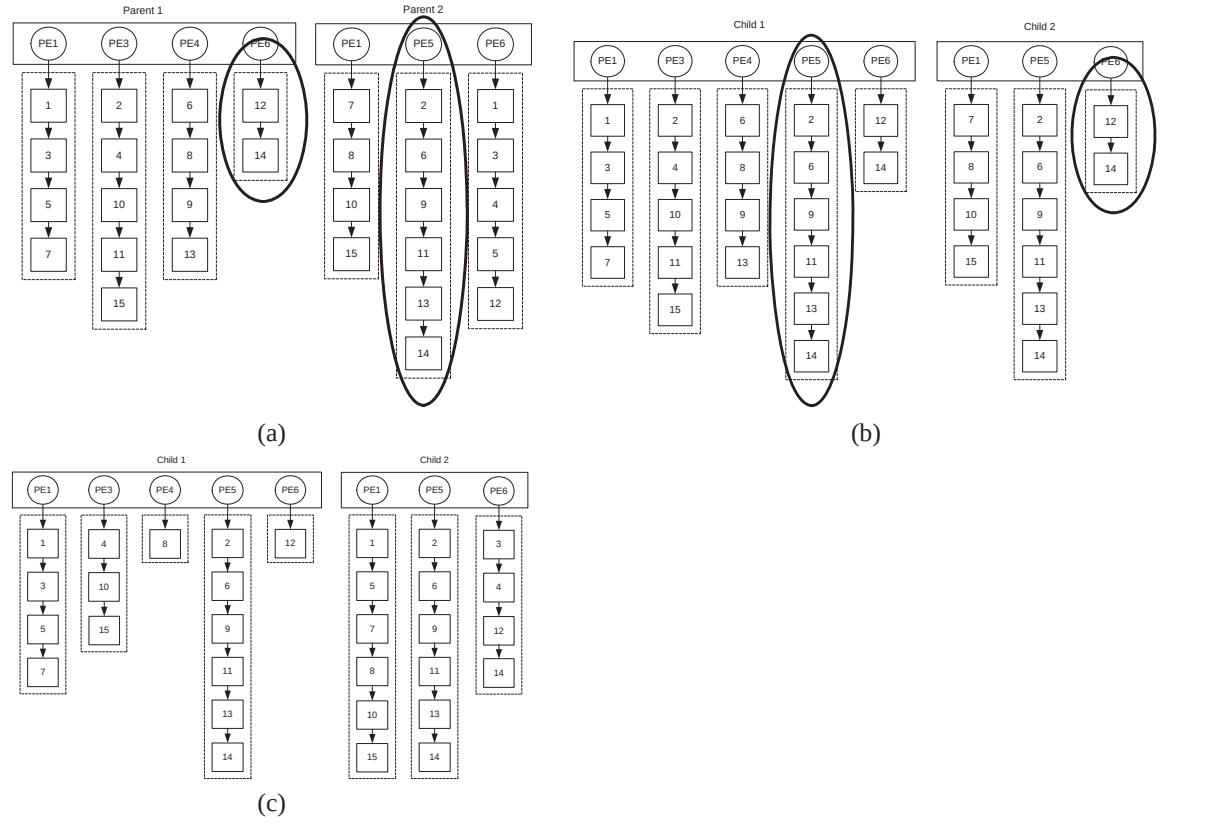

Fig. 2. Illustration of the PE schedule crossover for the various steps (a) Selection of random PE schedule, (b) Swapping of selected PE schedules, and (c) Deletion of duplicates and random insertion of missing tasks to form child chromosome

schedule or replaces the original schedule of that particular PE, if it is present. Duplicated tasks are deleted while missing tasks are randomly inserted to the other original PE schedules. The new PE schedule will remain intact. To ensure the feasibility of chromosomes after the crossover, the priority list computed at the beginning of the evolutionary process is used to sort the task assigned to each PE in ascending order to preserve feasibility.

D. Specialized Mutation

This paper applies three different specialized mutation operators to improve the diversity of evolving population.

- *Partial Exchange:* The partial exchange operation involves a number of partial schedule exchanges. For each exchange, two PE schedules are randomly chosen and a segment of the selected schedules is then randomly selected and exchanged. In addition, a mechanism is in place such that no PE schedules will not be selected twice in a particular partial exchange operation.
- *Schedule Merge:* This operation concatenates the two PE schedules with the least number of tasks in the chromosome. Intuitively, this operation is not applicable to solutions with only one PE schedule.
- *Schedule Split:* This operation first search for the PE schedules with the most number of tasks, and breaks the schedule into two at a random point. After which, the upper segment of the divided schedule is assigned randomly to either an idle PE or inserted into the PE schedule with the least number of tasks.

For every chromosome undergoing the mutation process, only one particular mutation operator is applied each time. Similar to the PE Schedule crossover, each PE schedule is sorted based on the priority list at the end of the mutation operation.

E. Local Search

1) Partial list scheduling: The optimality of the multiprocessor schedule is only as good as the last completion time of the task. The idea of partial list scheduling (PLS) is to split up the workload among the PEs with the best and worst completion times to improve the makespan. The first step in this heuristic is to select the appropriate PEs from which all tasks are extracted and placed in a list. These PEs are selected based on two criteria, either the PE has a completion time that is greater than the upper quartile or its completion time is lower than the lower quartile of the PE completion times. In the next step, the extracted tasks are sorted based on their priorities determined at the start of the evolutionary process. The tasks, in the order of their priorities, are then assigned to the best possible processor, i.e, the one which allows the earliest start time considering ITC. The new solution will be compared against the original and the better of the two will be retained.

2) Duplication scheduling: In multiprocessor scheduling with task interdependencies, some PEs will be idle during various time slots because some task require data from its parent tasks which are assigned to other processors. The idea of duplicating tasks in these idle time slots is to reduce the waiting and ITC delays incurred to reduce the makespan. The pseudocode of the duplication scheduling (DS) heuristic is shown in Fig. 3.

Duplication Scheduling Local Search

```
FOR   All PE Schedules
      FOR   All Task in PE Schedule
            Compute T_idle before task execution
            Determine parents of task
            Sort parents in descending order of completion time
            FOR   Parents
                  Determine T_exe required if duplicated
                  IF    Execution time< T_idle
                        Duplicate parent
                        Update T_idle: T_idle = T_idle - T_exe
                  ELSE
                        Break
                  END
            END
      END
END
Sort task based on priority
Evaluate new solution
IF    new solution is better than old solution
      Replace old solution
END
```

Fig. 3. Pseudocode of the Duplication Scheduling Local Search

The task duplication procedure is conducted iteratively every task in the order of its execution for each PE. The heuristic first determines the idle time which is the difference between the actual and earliest possible start time of the task. It then attempts to duplicate the parent tasks, in the order of their contribution to the delay, until the idle time is used up. The new solution will be compared against the original and the better of the two will be retained.

F. Algorithmic Flow

The algorithmic flow of the HEA is shown in Fig. 4. The optimization process begins with the initialization of the population based on the procedure described in Section III-A. After the initial evolving population is formed, all the chromosomes are evaluated and ranked according to their final execution time in the population. Following the ranking process, an archive population is updated. In this paper, an archive is applied to store all the best solutions found during the search. The archive maintains a fixed number of solutions and the updating process consists of a few steps. The evolving population and the archived solutions are first combined and all duplicate solutions are deleted. The remaining solutions in the combined population are then inserted into the archive in the order of increasing rank until the archive is filled.

The binary tournament selection scheme is then performed on the archive. Random pairs of individuals from the evolving archive are selected and from each pair, the chromosome with the lower rank is selected for reproduction. This procedure is performed until the mating pool is filled to preserve the original population size. The genetic operators consist of the PE schedule crossover and the three mutation operators

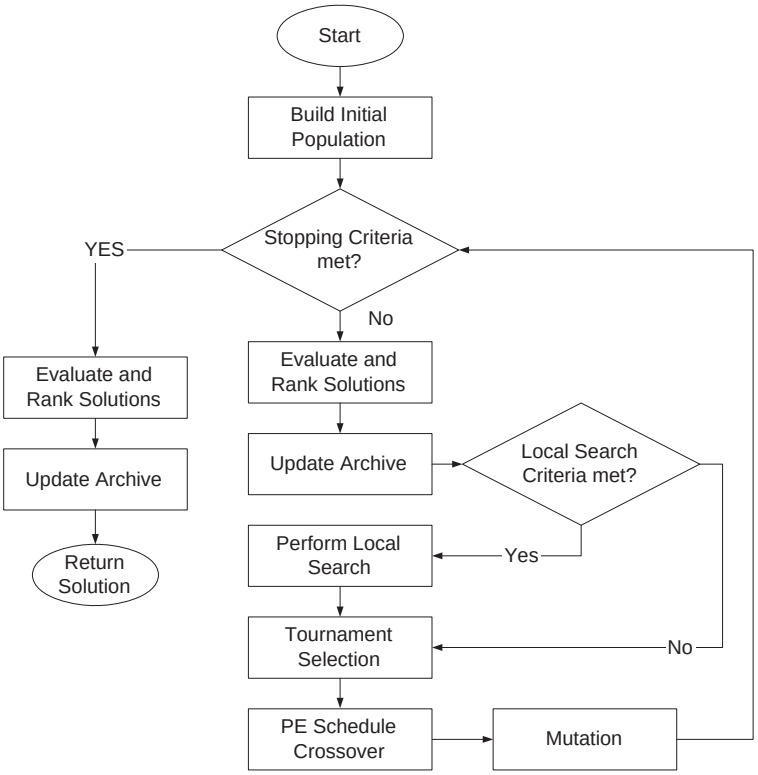

Fig. 4. Flowchart of HEA

presented in Section III-D and III-E respectively. The PLS and DLS are applied to the archive populations at a fixed interval, T_{LS}, for better local exploitation in the evolutionary search. Different schemes for incorporating the two local search methods will be explored in Section IV. The evolution process is repeated until the stopping criterion is satisfied.

IV. Simulation Results and Analysis

This section presents the extensive simulation results and analysis of the proposed HEA. The simulations are implemented using Matlab on an Intel Pentium 4 2.8 GHz computer and the results shown are based on the final makespan value of the best archived solution. Thirty independent runs are performed for each of the test functions in order to obtain the statistical information, such as consistency and robustness of the algorithms. The various parameter settings for the algorithm are listed in Table III. Section IV-A demonstrates the effectiveness of the proposed local search operators, as well as analyzes how the various settings of the local search heuristics will affect algorithmic performances. Section IV-B investigates the impact of different problem characteristics on HEA performances and how it compares against conventional heuristics.

TABLE III
Parameter setting for HEA

Parameter	Settings
Populations	Population size 20; Archive size 20.
Chromosome	Variable length chromosome;
Selection	Binary tournament selection
Crossover rate	0.9
Mutation rate	0.3
Evaluations	600
Local search frequency, T_{LS}	5

A. Effects of Local Search

The HEA incorporates the local search heuristics in order to exploit local schedules in parallel with global evolutionary optimization. In this section, the dynamics and parameter settings of PLS and DS are examined. Note that T1 and T4 are used in the study here since it has been observed in previous works that ITC will have severe impact on schedule optimality.

Six settings of HEA with various implementations of the local search operators are investigated as shown in Table IV. No local search is applied in setup 1 while only one heuristic is applied for each solution undergoing local search for setups 2-6. In setup 2, either PLS or DS is randomly applied. In the third and fourth setup, only one heuristic is applied. The asterisk (*) in setup 5 and setup 6 denotes which local search is activated first as they are alternately executed.

The evolutionary trends of the makespan averaged over 30 runs for T1 and T4 are plotted in Fig. 5(a)-(b). From the plots, it can be observed that the application of local search results in significant dips in the convergence trace, particularly in instances where DLS is applied. Fig. 5(a)-(b) distinctively demonstrate the effectiveness of local exploitation in the HEA as the five setups which incorporates local search performed

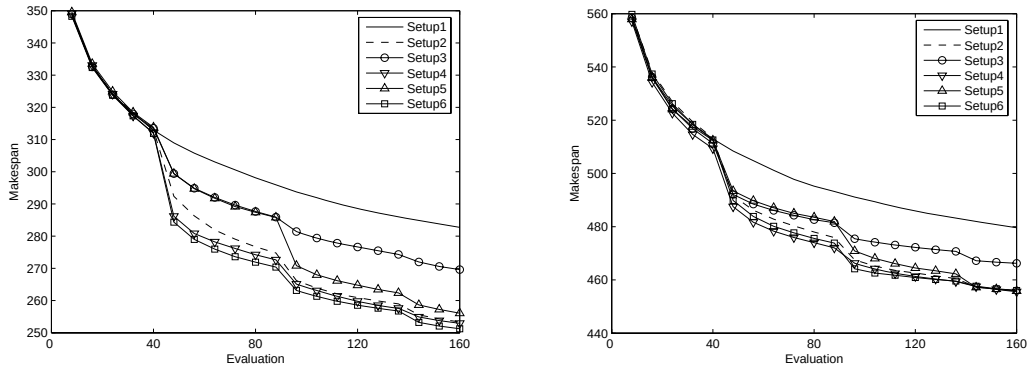

Fig. 5. Evolutionary trend of the six setups for (a) T1 and (b) T4

TABLE IV
DIFFERENT CASE SETUPS TO EXAMINE CONTRIBUTION OF THE LOCAL SEARCH HEURISTICS.

	1	2	3	4	5	6
PLS	-	Random	Yes	-	Alternate*	Alternate
DLS	-	Random	-	yes	Alternate	Alternate*

better as compared to setup 1. The performances of setup 2, setup 3, setup 5, and setup 6 are comparable, although the combination of DLS being activated first and PLS in setup 6 seems to have a slight edge for both problems. On the other hand, setup 5 which activates PLS first has a slower convergence rate for both T1 and T4.

The effectiveness of duplicating tasks in reducing overall completion time is also evident since the four settings of setup 2, setup 3, setup 5 and setup 6, are able to find solutions with makespans that are significantly lower than those found without local search and by PLS only. Interestingly, the application of DLS seems to have more impact on T1 with an average of 10% improvement as compared to 5% for T4 which has a more severe CCR restriction. Setup 6 will be used as the default setup for all subsequent experiments.

B. Investigation of Other Test Problems

In order to examine the effectiveness of HEA, a comparative study with conventional LSH and DSH is carried out based upon the 10 benchmark problems described earlier. As before, 30 simulation runs are conducted for all test problems and the results are summarized in Table V. LSH and DSH are deterministic heuristics and only one solution is produced for each problem.

As noted before in Section IV-A, the effectiveness of task duplication is evident by comparing the performances between LSH and DSH. The difference between the two conventional heuristics becomes even more apparent as the CCR or degree of heterogeneity increases. On the other hand, the HEA outperforms both heuristics for all test problems. With the exception of T1, it can be observed from Table V that the third quartile makespan value attained by HEA is much lower as compared to LSH and DSH for the benchmark problems. This also implies that the HEA is capable of evolving good schedules consistently.

V. CONCLUSION

In this paper, we proposed a hybrid evolutionary algorithm (HEA) specifically designed to solve the HMPSP by means of a variable-length chromosome, as well as specialized genetic and local search operators. The starting population is initialized using a random list scheduling heuristic to preserve the precedence relationships between the tasks. The evolutionary process is driven two primary variation operator – a schedule crossover and three variants of the mutation operator – partial exchange, schedule merge, and schedule split; the local search operators on the other hand consists of a partial list scheduling and duplication scheduling approach. Our results showed that the proposed genetic operators, when coupled with the local search operators performed better than in the case where any one of the operators were omitted. For future work, we would like to examine, in more detail, the performance of these scheduling algorithms for larger-sized problems, and if given the opportunity, implement them on actual multiprocessor systems.

REFERENCES

[1] I. Ahmad and Y. K. Kwok, "Optimal and near-optimal allocation of precedence-constrained tasks to parallel processors: Defying the high complexity using effective search techniques," in *Proceedings of 1998 International Conference on Parallel Processing*, pp. 423-431, 1998.

[2] I. Ahmad and Y. K. Kwok,"On Exploiting Task Duplication in Parallel Program Scheduling," IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 9, pp.872-892, 1998.

[3] S. Baskiyar and C. Dickinson, "Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication," *Journal of Parallel and Distributed Computing*, vol. 65, no. 8, pp. 911-921, 2005.

[4] T. Blickle, J. Teich and L. Thiele, "System Level Synthesis Using Evolutionary Algorithms," TIK-Report, Nr. 16, 1996.

[5] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810837, 2001.

TABLE V

SIMULATION RESULTS OF LSH, DSH AND HEA FOR THE VARIOUS BENCHMARK PROBLEMS.

	LSH	DSH	HEA		
			1st quartile	Median	3rd quartile
T1	241.2088	238.4844	236.5115	236.5379	238.6333
T2	339.3681	333.3969	313.3463	315.4760	317.0450
T3	438.3704	400.8312	368.8242	371.2755	372.9756
T4	496.6009	473.6011	440.6800	442.7701	443.913
T5	301.7023	299.9843	293.0909	295.1409	297.8713
T6	342.1177	340.6002	319.5344	321.5004	323.6641
T7	350.9543	307.5352	295.8357	298.9078	304.0759
T8	322.0563	316.2526	275.6143	281.5325	284.9465
T9	388.6536	371.4102	337.1911	342.0356	344.8998
T10	454.2243	415.2846	391.3507	395.1606	397.0869

[6] E. K. Burke, P. Cowling and P. De Causmaecker, "A memetic approach to the nurse rostering problem,"*Applied Intelligence*, vol. 15, no. 3, pp. 199-214, 2001.

[7] P. E. Coll, C. C. Ribeiro, and C. C. de Sousa, "Test instances for scheduling unrelated processors under precedence constraints, " *http://www-di.inf.pucrio.br/celso/grupo/readme.ps*, 2002.

[8] R. C. Correa, A. Ferreira and P. Rebreyend, "Scheduling Multiprocessor Tasks with Genetic Algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 8, pp. 825-837, 1999.

[9] H. El-Rewini, T. G. Lewis, and H. H. Ali, Task Scheduling in Parallel and Distributed Systems Prentice Hall, 1994.

[10] T. Davidovic and T. G. Crainic, "New benchmarks for static task scheduling on homogenous multiprocessor systems with communication delays, " Publication CRT, 2003-04, Centre de Recherche sur les Transports, Universite de Montreal, pages 123-136, 2003.

[11] M.K. Dhodi, EH. Hielscher, R.H. Storer and J. Bhasker,"Datapath Synthesis Using aProblem Space Genetic Algorithm," *IEEE Transactions on CAD*, vol. 14, no. 8, 1995.

[12] P. M Franca, A. Mendes and P. Moscato, "A memetic algorithm for the total tardiness single machine scheduling problem,"*European Journal Of Operational Research*, vol. 132, no. 1, pp. 224-242, 2001.

[13] M.R. Garey and D.S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., 1979.

[14] N. G. Hall and M. E. Posner, "Generating experimental data for computational testing with machine scheduling applications," *Operations Research*, vol. 49, pp. 854-865, 2001.

[15] E. S. Hou, N. Ansari and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113-120, 1994.

[16] H. Ishibuchi, T. Yoshida and T. Murata, "Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling,"*IEEE Transactions on Evolutionary Computation*, vol. 7 no. 2, pp. 204-223, 2003

[17] H. Kasahara and S. Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing," *IEEE Transactions on Computers*, vol. 33, no. 11, pp. 1,023-1,029, 1984.

[18] B. Kruatrachue and T.G. Lewis, "Duplication Scheduling Heuristic, a New Precedence Task Scheduler for Parallel Systems," Technical Report 87-60-3, Oregon State University, 1987.

[19] Y. Kwok and I. Ahmad, "Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm," *Journal of Parallel and Distributed Computing*, vol. 47, no. 1, pp. 58-77, 1997.

[20] Y. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406-471, 1999.

[21] T.G. Lewis and H. El-Rewini, Introduction to Parallel Computing. New York: Prentice Hall, 1992.

[22] B.S. Macey and A.Y. Zomaya, "A Performance Evaluation of CP List Scheduling Heuristics for Communication Intensive Task Graphs," in *Proceedings of the Joint 12th International Parallel Processing Symposium and Ninth Symposium on Parallel and Distributed Programming*, pp. 538-541, 1998.

[23] P. Merz and B. Freisleben, "Fitness landscape analysis and memetic algorithms for the quadratic assignment problem," *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 4, pp. 337-352, 2000.

[24] Y. S. Ong and A. J. Keane, "Meta-Lamarckian learning in memetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 99-110, 2004

[25] Y. S. Ong, M. H. Lim, N. Zhu, and K. W. Wong, "Classification of Adaptive Memetic Algorithms: A Comparative Study," *IEEE Transactions On Systems, Man and Cybernetics - Part B*, vol. 36, no. 1, pp. 141-152, 2006.

[26] C. Papadimitriou and M. Yannakakis, "Toward an Architecture Independent Analysis of Parallel Algorithms," SIAM J. Computing, vol. 19, pp. 322-328, 1990.

[27] G. Ritchie and J. Levine,"A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments," in *Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group*, 2004.

[28] T. Tsuchiya, T. Osada, and T. Kikuno, "Genetic-Based Multiprocessor Scheduling Using Task Duplication," *Microprocessors and Microsystems*, vol. 22, pp. 197-207, 1998.

[29] A. S. Wu, H. Yu, S. Jin, K. C. Lin and G. Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 824-834, 2004.

[30] Y. W. Zhong, J. G. Yang, H. N. Qi, "A Hybrid Genetic Algorithm for Task Scheduling in Heterogeneous Computing Systems," in *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, pp. 2463-2468, 2004.

[31] A. Y. Zomaya, C. Ward and B. Macey, "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 8, pp. 795-812, 1999.