Finding Near-Optimal Task Scheduling for Distributed Real-Time Environments

Abstract—**Real-world applications are composed of multiple subtasks which have data dependencies in general. To exploit distributed processing platforms, subtask matching and scheduling, which consist of assigning subtasks to machines and ordering inter-machine data transfers, plays a vital role. However, optimal scheduling subtasks to machines and finding optimized network topology is an NP-complete problem. The problem will be more complicated when the subtasks have real-time deadlines for termination. Exploring the whole search space in order to find the optimal solution is not feasible in a reasonable amount of time, therefore meta-heuristics are mostly used to find a near-optimal solution. We propose a multi-population evolutionary approach for near-optimal scheduling and topology optimization that guarantees end-to-end deadlines of subtasks in distributed processing environments. In this paper, two different exploration scenarios including single and multi-objective exploration have been analyzed. The main aim of single objective exploration algorithm is to achieve the minimal number of processing machines for all the subtasks, where a multi-objective optimization tries to optimize two objectives simultaneously consisting the total number of processing machines and end-to-end finishing time for all the jobs. The potential of the proposed approach is demonstrated by experiments based on an industrial automation use case for mapping a number of jobs, each of which consists of a number of tasks to a distributed environment.**

Index Terms—**Distributed Task Scheduling, Real-Time Processing, Evolutionary Computing, Topology Optimization**

I. Introduction

Industrial applications often require guaranteeing real-time execution, fault tolerant implementations and providing reliable functionality. In general, it is impossible for a single processing machine to satisfy all these needs. However, a distributed processing environment provides a variety of computational capabilities, which can be utilized to perform an application that has diverse execution requirements. An application job can be decomposed into subtasks. Subtasks may have data dependences and it is possible that each subtask needs a certain computational throughput. For distributing subtasks, the following decisions should be made respectively: ① subtasks matching, i.e. assigning subtasks to processing machines, and ② subtasks scheduling, i.e. defining subtask execution order and the order of data transfers among machines. The general goal of subtasks matching and scheduling is to minimize the end-to-end cost of computation, i.e. minimizing overall response time of the application, minimizing the number of processing machines, or both.

Performance of such parallel systems can be optimized by employing an efficient task matching and scheduling approach, however, the matching and scheduling problem is an NP-complete problem [1]. Using exhaustive approaches for finding optimal solution is time-consuming and is impossible in practice. Many heuristic task scheduling strategies have been proposed [2], [3] to find a near-optimal solution in a resealable amount of time. Evolutionary Computing (EC) is a set of methods proposed to solve the matching and scheduling problem. Genetic Algorithm (GA) is a popular EC method which can better locate a near-optimal solution than other similar approaches in most cases [4]–[7]. Although GA is a powerful solution, defining a proper fitness function is always challenging and requiring expertise especially when the size of design space is huge. Plus, GA is relatively slow and may be trapped in local optima.

To overcome aforementioned challenges, Multi-Population Genetic Algorithm (MPGA) [8] is leveraged in this research for subtask matching and scheduling in a collection of nonuniform processing machines. MPGA is a static scheduling strategy, where the execution times of subtasks and the data transfer times between subtasks are known. MPGA is the parallel version of GA that provides better convergency rate and more speedup compared to single population GA. In addition, MPGA highly reduces the probability of falling into local optima trap. Two different MPGA strategies have been considered to solve the matching and scheduling problem including single objective and multi-objective optimizations. While the single objective optimization minimizes the number of processing machines, the multi-objective optimization considers the second metric, jobs total finishing time, to find solutions satisfying multiple user needs.

In nutshell our main contributions are:

- Presenting a MPGA optimization method for solving subtask matching and scheduling problem in a distributed environment.
- Defining a novel fitness function to efficiently explore the design space in both single and multi-objective optimizations.
- The evaluation results based on an industrial use case shows the impact of the proposed fitness function while converging to better solutions.

The paper is organized as follows. Section II defines the matching and scheduling problem and the industrial use case. Section III explains the GA and the specifications of fitness functions for both single objective and multi-objective optimizations. Section IV presents the experimental results and demonstrates the efficiency and convergency of the proposed algorithm. Some related work reviewed in Section V. We end with concluding remarks and future work in Section VI.

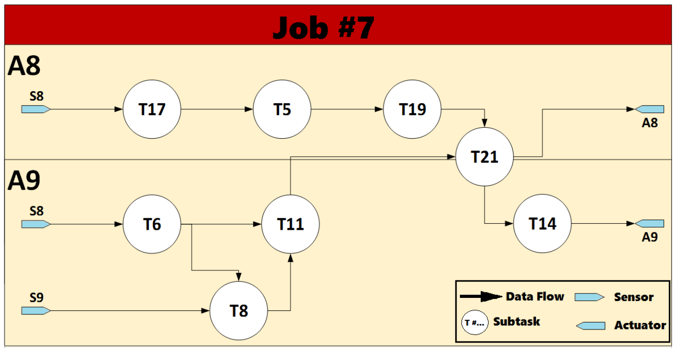


Fig. 1. Representing a Job Sample From the Industrial Use Case (Job #7).

II. Problem Definition

A. Problem Assumption

To only focus on the matching and scheduling problem, we made the following assumptions. First of all, we assumed that each subtask is written in a machine-independent language. Moreover, it is assumed that an application job is decomposed into multiple subtasks and we know all the data dependencies among subtasks before execution. The load complexity of all the subtasks and their execution time on each processing machine is known a priori. It is assumed that for each subtask, there is a couple of input nodes that produces raw input data (*sensors*) and there are some output nodes which consume subtasks processing results (*actuators*). Obviously the input of subtask is coming from sensors or the output of other subtasks and similarly the output results of each subtask will be consumed by actuators or other subtasks. The distributed processing platform is nonuniform that consists of multiple homogeneous machines with various processing potential. All the processes on processing machines are non-preemptive meaning each processing machine completes the current task before calling the next task.

Also, all input data items of a subtask must be received before its execution can begin, and none of its output data items is available until the execution of this subtask is finished. If a data conditional is based on input data, it is assumed to be contained inside a subtask. A loop that uses an input data item to determine one or both of its bounds is also assumed to be contained inside a subtask. When two communicating tasks are mapped onto the same processing machines we assume that the communication delay is zero. However, when they are mapped onto different processors a finite communication delay is assumed and modeled. Fig. 1 shows the dependency graph between subtasks of a job example from the industrial use case (see Section II.A).

B. Industrial Use Case

@Cristina: I also attached Fig 2, job table

III. MPGA Description

GA is an iterative population-based exploration solution mimicking the process of natural selection and evolution where the characteristics of the process can be utilized in solving optimization problems. All GA-based methods have an initial population where selection, crossover, mutation operators are applied to initial population for producing improved population. The operations will be repeated until satisfying user criteria (reaching suitable results) or meeting predefined number of iterations. The following subsections explain the basic components of GA.

*a) **(Step 0) Generating Initial Population**:* The initial population includes random solutions in the design space, where each solution represented by chromosome is a scheduling for all the jobs. The size of initial population depends on the size of design space. To check the validity of solutions in the initial population, each solution is examined by using the Equation (1) objective function.

*b) **(Step 1) Fitness Evaluation**:* Objective function (fitness function) is a metric for comparing different scheduling that satisfy problem constraints. Equation (1) and Equation (2) represent the fitness functions for single-objective and multi-objective optimizations, respectively.

$$Fitness\_1 = \#Processors + (\gamma \times (\alpha + \beta + \theta)) \tag{1}$$

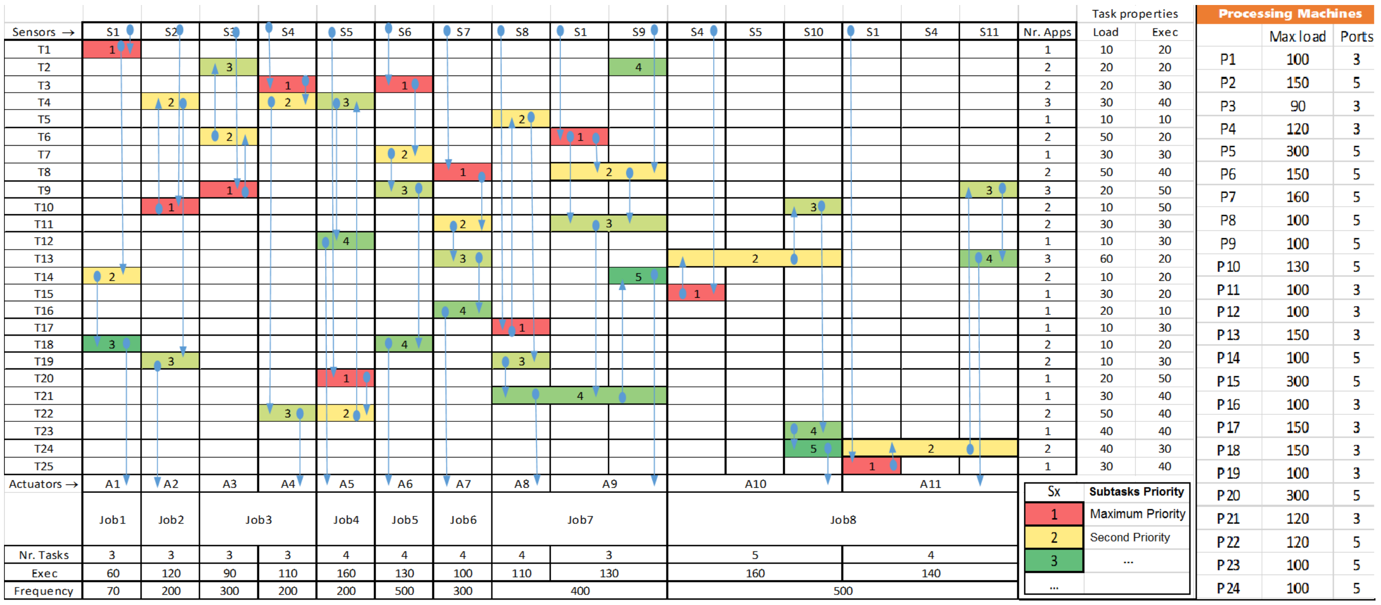
$$Fitness\_2 = \frac{\#Processors}{\gamma} + \frac{Runtime}{BiggestDeadline} + 3 \times (\alpha + \beta + \theta) \tag{2}$$

α is the total extra loads of all subtasks that exceed the load of processing machines, β is the total extra deadline of all subtasks that exceed the real-time deadlines, θ is the total extra ports of all job assignments that exceed the number ports pf processing machines, and γ is equal to 23, the total number of processing machines. $BiggestDeadline$ is the maximum possible time fo the slowest subtask.

*c) **(Step 2) Selection**:* Obviously the schedules with better fittest function are selected as the next population and the others will be removed from population set. The goal is to find a solution in design space with lowest fitness function.

*d) **(Step 3) Crossover Operators**:* is the most important operator of GA. GA randomly selects two genomes from the population set based on a certain crossover rate. Then two genome strings exchange parts of their corresponding chromosomes to create two new genomes. In the use case example the chosen jobs scheduling are exchanged with the other scheduling on corresponding processor for producing two new schedules with most likely better schedules. Fig. 3 illustrates the representation of the all jobs scheduling by a genome type. Each genome consists of 42 portions since the use case has 42 different subtasks. All possible assignments to processing machines for each subtask which is equal to 23 (γ).

*e) **(Step 4) Mutation Operator**:* The main goal of mutation operator is to increase genetic diversity. Mutation alters one gene value (assigned processor to subtask) in a chromosome string from its initial state. The solution may be better or even worst solution by using mutation. Mutation



Task	Nr. Apps	Task properties Load	Task properties Exec
T1	1	10	20
T2	2	20	20
T3	2	20	30
T4	3	30	40
T5	1	10	10
T6	2	50	20
T7	1	30	30
T8	2	50	40
T9	3	20	50
T10	2	10	50
T11	2	30	30
T12	1	10	30
T13	3	60	20
T14	2	10	20
T15	1	30	20
T16	1	20	10
T17	1	10	30
T18	2	10	20
T19	2	10	30
T20	1	20	50
T21	1	30	40
T22	2	50	40
T23	1	40	40
T24	2	40	30
T25	1	30	40

Actuators →	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11
	Job1	Job2	Job3		Job4	Job5	Job6	Job7		Job8	
Nr. Tasks	3	3	3	3	4	4	4	4	3	5	4
Exec	60	120	90	110	160	130	100	110	130	160	140
Frequency	70	200	300	200	200	500	300	400		500	

Sx	Subtasks Priority
🟥 1	Maximum Priority
🟨 2	Second Priority
🟩 3	...
...	

Processing Machines	Max load	Ports
P1	100	3
P2	150	5
P3	90	3
P4	120	3
P5	300	5
P6	150	5
P7	160	5
P8	100	5
P9	100	5
P10	130	5
P11	100	3
P12	100	3
P13	150	3
P14	100	5
P15	300	5
P16	100	3
P17	150	3
P18	150	3
P19	100	3
P20	300	5
P21	120	3
P22	120	5
P23	100	5
P24	100	5

Fig. 2. Representing The industrial Use Case Including Jobs, Intra Subtasks Subtasks Dependencies, Subtasks Load Complexity and Real-time Deadlines, and Processing Machines Specifications.

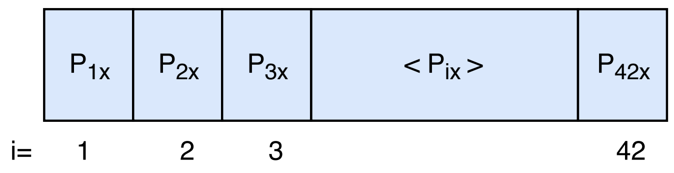


Fig. 3. Representing a Valid Matching and Scheduling by GA/MPGA Chromosome Type.

forces GA to get rid of local optima. For doing mutation, we need to randomly select one gene in chromosome and modify its assigned value to a new valid number.

After each cycle of selection, crossover and mutation, the newly generated set of solutions (schedules) is called as new generation. Every generation is evaluated based on the fitness function to determine if they represent a good enough solution to satisfy the fitness function. This determines if the GA can stop searching, or if otherwise, for the GA to continue searching until the set stopping criteria is met. The stopping criteria could be the number of generations, or evolution time, or fitness threshold, or fitness convergence, or population convergence. In our case, the number of generations was set as the stopping criteria. The schedule obtained after the stopping criteria will be the optimal or near optimal schedule.

A. MPGA Algorithm

Although EC methods can improve the quality of results, using them have some difficulties. First of all, an evolutionary algorithm may not converge towards the optimal solutions or even to near-optimal solutions in the case of very huge exploration space. One possible solution is to increase the initial population size, but leading to increase the execution time of evolutionary algorithms. Parallelizing these algorithms can remarkably diminish their execution time and improve the quality of results. In the parallelized GA, multiple processors work together where each one runs a simple GA and has an independent populations. After a predefined number of iterations, all processors share their best chromosomes among each other (***migration operation***). The whole parallel procedure is called MPGA. Sharing the best individuals aids the MPGA to get avoid of local optima. Fig. 4 represents the behavior of the MPGA and the flowchart of consequent operations is shown in Fig. 5. The pseudocode of MPGA is presented in Algorithm 1.

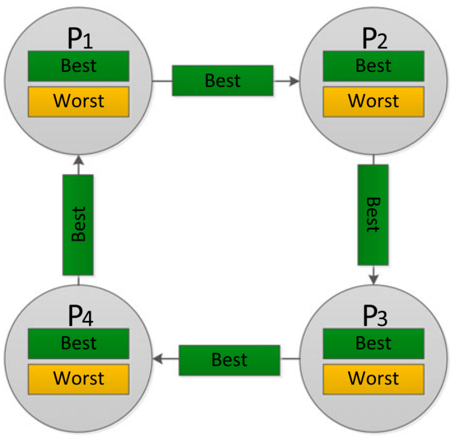


Fig. 4. Multi-population Migration Operation.

The inputs of proposed meta-heuristic optimization approach include: ① the specification of processing machines

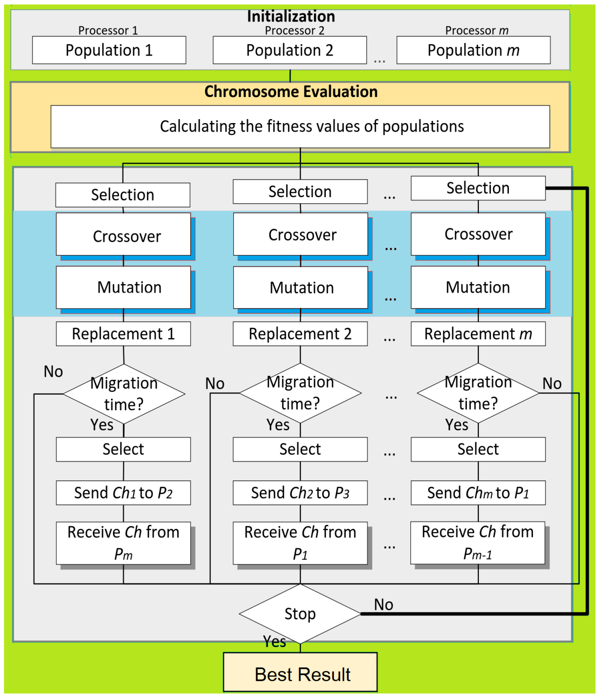


Fig. 5. Flowchart of MPGA.

including maximum processing potential, the total number of input/output ports, and ② the specifications of jobs and subtasks including load complexity, run-time deadlines, and subtask dependencies.

Algorithm 1: Pseudo Code of MPGA

```
Input: • Processor $P_i$: $1 \leq i \leq$ # Processors
• Distributed Processing Machines Specifications
• Jobs and related Subtasks Specifications
• $N$: Population Size
• $T$: Maximum Number of Iterations
Output: A Set of Near-Optimal Solutions
Function MPGA($N$, $T$):
    (Step 0): $U_{i,0}$= Random_Population (N);
      //Creating initial random population and assign to each $P_i$
      (Step 1): Fitness_Function ($U_{i,0}$); //Evaluating the objectives of each solution in the all populations
      $t = 1$;
      while $t \leq T \mid SatisfyingUserNeeds$ do
          (Step 2): $U'_{i,t}$ = Select ($U_{i,t}$); //Select some chromosomes from the $U_{i,t}$ randomly.
          (Step 3): $U''_{i,t}$ = Crossover ($U'_{i,t}$)
          (Step 4): $Y_{i,t+1}$ = Mutation ($U''_{i,t}$)
          (Step 1): Fitness_Function ($Y_{i,t+1}$);
          (Step 5): if $Iterations\%MigrationGap == 0$ then
              Select the best chromosome from $Y_{i,t+1}$
               Send the best chromosome to $P_{i+1}$
               Receive the best chromosome from $P_{i-1}$
          $t = t + 1$;
    return $Y_{i,t+1}$
```

IV. Evaluations

This section presents the results of experiments that have been fulfilled to verify the impact of the proposed MPGA on the industrial use case. The evaluations have been done based on two different optimization strategies including single objective and multi-objective optimizations.

A. *Implementation Detail*

MPGA is implemented in C++ and MPI library has been utilized for parallelization. Ring topology is used for connections between processors for running MPGA. For the implementations, an Intel Core i7-4770 CPU 3.40 GHz with 16.0 GB RAM running on 64-bit Windows 10 has been used. Seven cores have been leveraged in the parallel implementation. The specification of MPGA parameters is shown in Table I.

TABLE I
MPGA Algorithm Parameters.

Parameter	Value
N: Initial Population Size (Each Processor)	100
# Populations	15
Total # Iterations	750
Crossover Rate	
Mutation	One-Point Mutation
Migration Rate	3
Migration Gap	25

B. *Experimental Results Convergency*

One of the main limitations of evolutionary algorithms is decreasing the convergence speed by increasing the number of iterations leading to make non-convergent results in low iterations for difficult problems. Fig. 6 and Fig. 7 represent the convergency of fitness functions for both single and multi-objective, respectively. It can be easily observed from the convergency figures that both strategies are highly convergent toward the optimal results by contentious reduction in fitness functions (se Equation (1) and Equation (2)).

a) Single Objective Optimization: Fig. 8 illustrates the variation trend of total number of utilized processing machines by increasing the number of iterations. As mentioned before, the aim of single objective optimization is to decrease the number of processing machines used in jobs scheduling. Fig. 8 shows considerable improvement in finding scheduling with less required processing machines. According the results of Table II, we need 22 processing machines for scheduling in the first iteration, while by proceeding the exploration algorithm, we found a solution with only seven required processing machines. Although there exist some breaks in continuous improvement, the overall trend moves toward improvement.

b) Multi-Objective Optimization: As mentioned before, the total number of processing machines and end-to-end runtime for all the jobs are the two main objectives of MPGA.

TABLE II
EXPERIMENTAL RESULTS COMPARED TO ...

Exploration Approach	**Runtime**	**# Processing Machines**
Single Objective Equation(1)	250	7
Multi-Objective Equation(2)	Solution①: 210 Solution②: 250 Solution③: 160	7 8 9

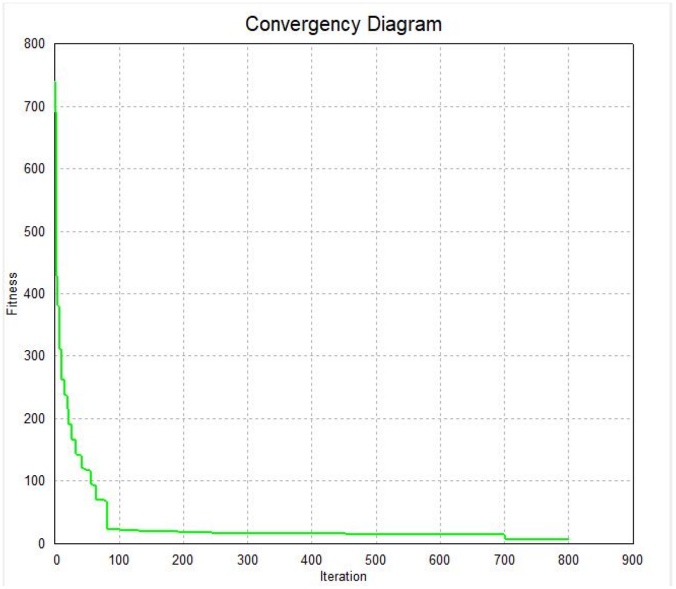


Fig. 6. Convergency Diagram of Single Objective Optimization (# Processing Machines).

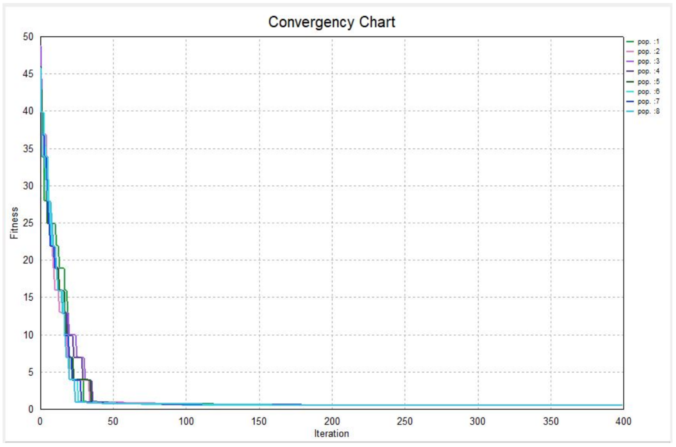


Fig. 7. Convergency Diagram of MPGA (# Processing Machines, End-to-End runtime).

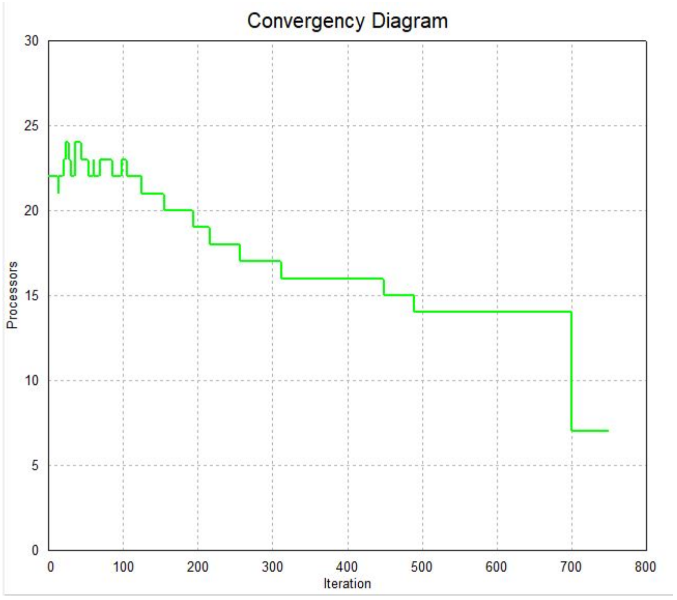


Fig. 8. Convergency Diagram of Single Objective Optimization.

Fig. 9 and Fig. 10 illustrate the convergency figures of required processing machines for scheduling and end-to-end runtime for all the schedule jobs, respectively. We can conclude from the figures that both the objectives are approaching toward optimized results. Although there are some failures or stops in achieving better results in each iteration, the overall Progression of MPGA always approaches toward superior outcomes (Fig. 11).

Table II shows three different solutions on the Pareto frontier of the last Population. We have a variety of options based on the user needs. Solution① is an scheduling with minimized number of processing machines (7 processing machines) while takes more time, 210 time unit, for running. On the other hand, Solution③ provide the minimum elapsed end-to-end runtime (160 time unit), while needs 9 processing machines for running.

C. Comparison Between MPGA and simple population GA

For evaluating the impact of multi population optimization on the matching and scheduling problem, The results of single objective optimization has been achieved by leveraging single population scheme. On the other hand, the results of multi-objective optimization has been achieved by using MPGA. We compared MPGA and simple GA schemes in terms of exploration time and quality of results in the following sections.

a) Exploration Time and Speedup: Fig. 9 and Fig. 2 represent the convergency of processing machines for MPGA and single population GA, respectively. MPGA achieve the best result after 400 iterations, while single population GA needs 750 iterations for finding the best solution. Obviously converging to the best result needs in less number of iterations by using MPGA which is the best proof to show the benefits of applying the MPGA, especially when the design space is large.

b) Quality of Results: According to the results of Table II, MPGA found a solution with 7 required processing machines and 210 time unit for total runtime execution, while single population GA found a solution with the same required processing machines but takes 250 time unit for total runtime.

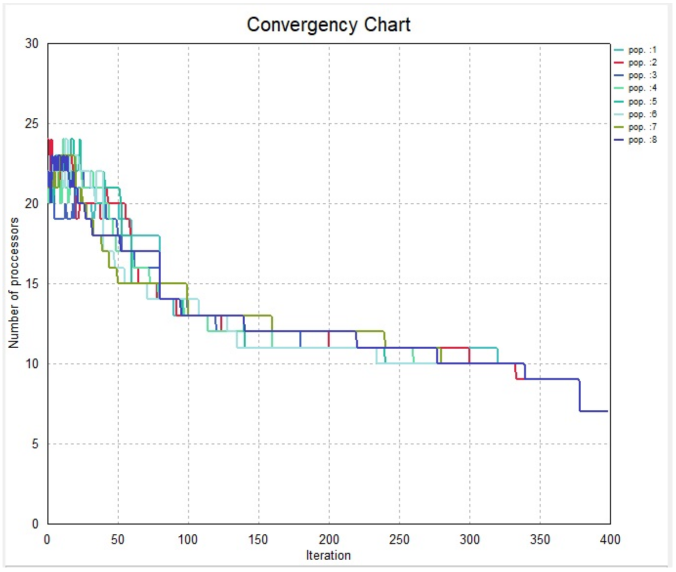


Fig. 9. Convergency Diagram of # Processing Machines in Multi-objective Optimization by Using MPGA Approach.

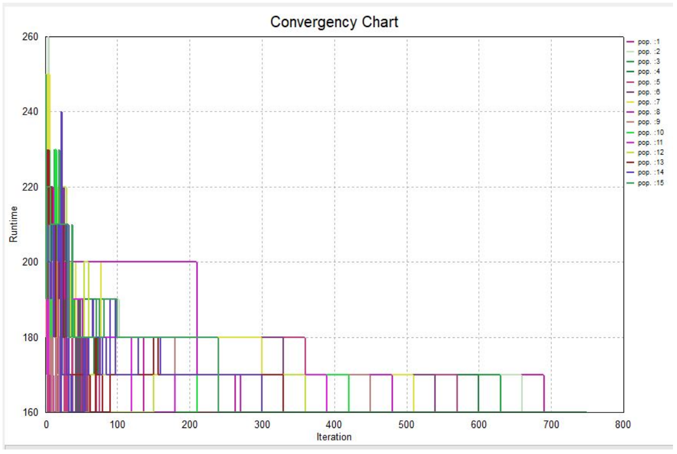


Fig. 10. Convergency Diagram of End-to-End runtime in Multi-objective Optimization by Using MPGA Approach.

MPGA provides more quality of results compared to single population GA even single population GA tries optimize only one objective.

D. Comparison Between MPGA and ...

E. Matching and Scheduling Results

Fig. 12 illustrates the best scheduling results for the studied use case after applying MPGA. Fig. 12.a represents a valid scheduling for all jobs and their related subtasks with minimum end-to-end runtime (Solution③). Fig. 12.b represents a valid scheduling for all jobs and their related subtasks with minimum number of processing machines (Solution①).

V. Related Work

Here, we first explore more traditional list scheduling heuristics that have considered communication costs.

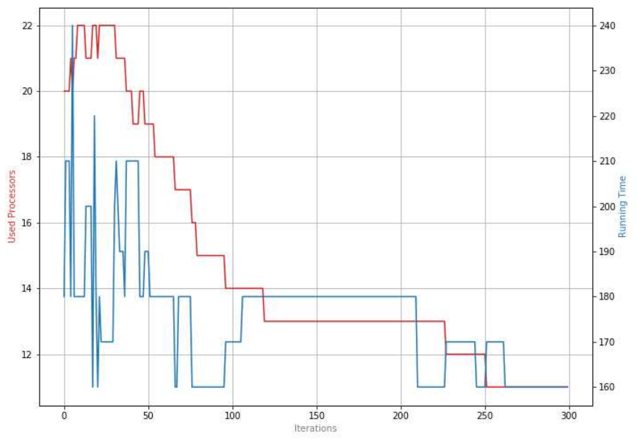


Fig. 11. Improvement Proceeding of Exploration Objectives Including The Number of Processing Machines and End-to-End Use Case Runtime.

The basic idea is to make an ordered list of nodes by assigning them orders, and then to repeatedly execute the following two steps until a valid schedule is obtained: ① Select from the list the node with the highest order for scheduling. ② Select a processor to accommodate this node. In realistic cases, scheduling needs to exploit parallelism by identifying the task graph structure and take into consideration task granularity, arbitrary computation, and communication costs.

In [9], the modified critical path algorithm (MCP) is proposed, based on the latest possible start time of a node. A node's latest possible start time is determined via the as-late-as-possible (ALAP) binding by traversing the task graph upward from the exit nodes to the entry nodes while pulling the node's start times downwards as much as possible. The latest possible start time of the node itself is followed by a decreasing order of the latest possible start times of its successor nodes. Furthermore, in [9], the dominant sequence clustering algorithm (DSC) is presented. It is based on the dominant sequence, which is essentially the critical path of the partially scheduled task. CP (the critical path of task graph) node is a ready node. If so, DSC schedules it to a processor allowing the minimum start time. Such a minimum start time may be achieved by rescheduling some of the node's predecessors to the same processor. If the highest CP node is not a ready node, DSC does not select it for scheduling. Instead, it chooses the highest node which lies on a path reaching the CP for scheduling. Moreover, also in [9], the mobility directed algorithm (MD) is presented. MD selects a node at each step based on relative mobility which is defined as the difference between a node's earliest start time and latest start time. Similar to the ALAP binding, the earliest possible start time is assigned to each node via the as-soon-as-possible (ASAP) binding. This is performed by traversing the task graph downward from the entry nodes to the exit nodes while pulling the nodes upward as much as possible. Moreover, relative mobility is obtained by dividing the mobility with the

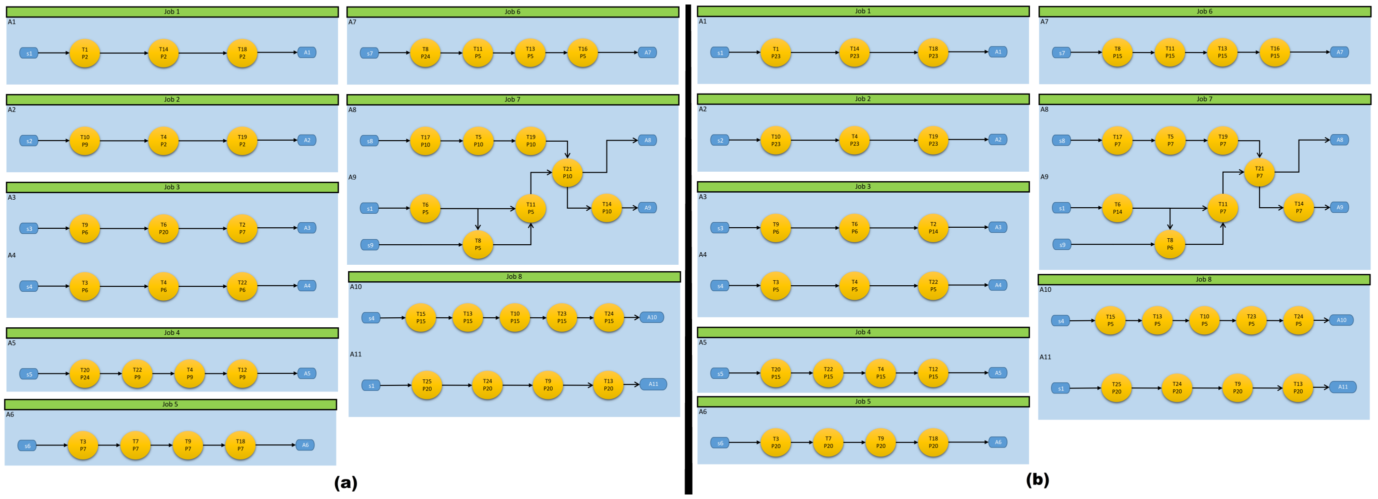


Fig. 12. (a): The Best Solution For Multi-Objective Optimization with Minimum End-to-End Runtime (Solution③). (b):The Best Solution For Multi-Objective Optimization with Minimum Number of Processing Machines (Solution①)

nodes computation cost. Basically, a node with zero mobility is a node on the CP. At each step, MD schedules the node with the smallest mobility to the first processor having a large enough time to accommodate the node without considering the minimization of the nodes start time. After a node has been scheduled, the relative mobility values of the remaining nodes are updated.

In [10], a MPGA is presented which outperforms deterministic and nondeterministic methods described in [11], [12]. In [13], a new encoding mechanism with a multi-functional chromosome is presented, using a priority representation that is called priority- based multi-chromosome (PMC). PMC can efficiently represent a task schedule and assign tasks to processors. It is another meta-heuristic method that uses a GA to achieve near-optimal scheduling of tasks.

Research on static mapping methods includes the work of Lei et al., who proposed a genetic mapping algorithm to optimize application execution time [14]. In their work, graphs represent applications and the target architecture is a NoC. Wu, et al. also investigated genetic mapping algorithms [15]. By combining dynamic voltage scaling techniques with mapping, they achieved 51% savings in energy consumption. Murali et al. explored mappings for more than one application in NoC design, using the tabu search (TS) algorithm [16]. Manolache, et al. investigated task mapping in NoCs, trying to guarantee packet latency [17]. For this purpose, both the task-mapping algorithm (TS) and the routing algorithm are defined at design time. Hu et al. presented a branch-and-bound algorithm to map a set of IP cores (IPs) onto a NoC with bandwidth reservation [18]. Their results show energy savings of 51.7% in the communication architecture. Marcon et al. investigated how to map modules into a NoC, targeting low energy consumption [19]. They compared several algorithms, using a model that characterizes applications by their inter-task communication volume. Xu et al. In [20] presented a task scheduling scheme on heterogeneous computing systems using a multiple priority queues genetic algorithm (MPQGA). Their experimental results for large-sized problems for a large set of randomly generated graphs as well as graphs of real-world problems with various characteristics showed that the proposed MPQGA algorithm outperformed two non-evolutionary heuristics and a random search method in terms of schedule quality.

VI. Conclusion and Future Work

Leveraging a distributed environment for task scheduling can enhance reliably and provide a fault tolerant processing scheme. However, there are some difficulties in distributing application jobs and scheduling them among processing platforms and usually leads this applications to suffer from inefficient performance. The problem will be more highlighted when we need to deal with subtask dependencies, nonuniform processing environment and requiring to guarantee real-time execution. In this paper, a parallel Multi-Population Genetic Algorithm (MPGA) is leveraged to overcome aforementioned barriers. For the evaluations, an industrial use case has been studied. The final results offer a better resource efficiency while guaranteeing real-time execution. In addition, MPGA provides better efficiency compared to other cutting-edge evolutionary approaches.

References

[1] Freund, R. F. Optimal selection theory for superconcurrency. Proc. Supercomputing 89. IEEE Computer Society, Reno, NV, 1989, pp. 699703.

[2] Adam TL, Chandy KM, Dicksoni JR. A comparison of list schedules for parallel processing systems. Communications of the ACM 1974;17(12):68590.

[3] Wu MY, Gajski DD. Hypertool: a programming aid for message-passing systems. IEEE Transactions on Parallel and Distributed Systems 1990;1(3):33043.

[4] Hou ESH, Ansari N, Hong R. A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 1994;5(2):11320.

[5] Hwang RK, Gen M. Multiprocessor scheduling using genetic algorithm with priority-based coding. Proceedings of IEEE conference on electronics, information and systems; 2004.

[6] Wu AS, Yu H, Jin S, Lin K-C, Schiavone G. An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 2004;15(9):82434.

[7] Majd, Amin, et al. "NOMeS: Near-optimal meta-heuristic scheduling for MPSoCs." Computer Architecture and Digital Systems (CADS), 2017 19th International Symposium on. IEEE, 2017.

[8] Y. Chen, Y. Zhong, Automatic Path-oriented Test Data Generation Using a Multi-population Genetic, Proc. Fourth International Conference on Natural Computation, pp. 566 570, Oct 2008.

[9] Adam TL, Chandy KM, Dicksoni JR. A comparison of list schedules for parallel processing systems. Communications of the ACM 1974;17(12):68590.

[10] R. Moradi and D. Dal, A Multi-Population Based Parallel Genetic Algorithm for Multiprocessor Task Scheduling with Communication Costs, 2016 IEEE Symposium on Computers and Communication (ISCC).

[11] Wu MY, Gajski DD. Hypertool: a programming aid for message-passing systems. IEEE Transactions on Parallel and Distributed Systems 1990;1(3):33043.

[12] Hou ESH, Ansari N, Hong R. A genetic algorithm for multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems 1994;5(2):11320.

[13] R. Hwang, M. Gen and H. Katayama, A comparison of multiprocessor task scheduling algorithms with communication costs Computers & Operations Research, Vol. 35, pp. 976 993, ELSEVIER, 2008.

[14] T. Lei and S. Kumar, A Two-Step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture, Proc. Euromicro Symp. Digital System Design (DSD 03), IEEE Press, 2003, pp. 180-187.

[15] D. Wu, B. Al-Hashimi, and P. Eles, Scheduling and Mapping of Conditional Task Graphs for the Synthesis of Low Power Embedded Systems, Proc. Design, Automation and Test in Europe (DATE 03), IEEE CS Press, 2003, pp. 90-95.

[16] S. Murali and G. De Micheli, Bandwidth-Constrained Mapping of Cores onto NoC Architectures, Proc. Design, Automation and Test in Europe (DATE 04), IEEE CS Press, 2004, pp. 896-901.

[17] S. Manolache, P. Eles, and Z. Peng, Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on NoC, Proc. 42nd Annual Design Automation Conf. (DAC 05), ACM Press, 2005, pp. 266-269.

[18] J. Hu and R. Marculescu, Energy- and Performance- Aware Mapping for Regular NoC Architectures, IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. 24, no. 4, 2005, pp. 551-562.

[19] C. Marcon et al., Comparison of NoC Mapping Algorithms Targeting Low Energy Consumption, IET Computers & Digital Techniques, vol. 2, no. 6, 2008, pp. 471-482.

[20] Y. Xu, K. Li, J. Hu and K. li, A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues, Information Sciences, Vol.270, pp. 255-287,Elsevier,2014.