Received: October 15, 2019

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

A Heuristic Approach for Scheduling in Heterogeneous Distributed Embedded Systems

Sethakarn Prongnuch[1]* **Suchada Sitjongsataporn[1]** **Theerayod Wiangtong[2]**

[1]*The Electrical Engineering Graduate Program, Faculty of Engineering, Mahanakorn University of Technology, Bangkok, Thailand*
[2]*Department of Electrical Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand*
* Corresponding author's Email: sethakarn.pr@ieee.org

Abstract: This paper presents a heuristic approach for workflow scheduling in heterogeneous distributed embedded system (HDES). A genetic algorithm (GA) and ant colony optimization (ACO) modified with the greedy algorithm introduced to the system contains multiple heterogeneous embedded machines (HEMs) working as a cluster. Users can remotely access and utilize their computational power. The communications on different types of buses are taken into account to find an optimal solution. New meta-heuristic information based on forwarding dependency is proposed to build probability for ACO to generate task priorities. Besides, a greedy algorithm for machine allocation is incorporated to complete task scheduling. Experiments based on random task graphs running in the HEM cluster demonstrate the effectiveness of the modified greedy ant colony optimization algorithm which outperforms the others by 33% more result quality.

Keywords: Heterogeneous distributed embedded system, Network cluster, Meta-heuristic task scheduling, Genetic algorithm, Ant colony optimization, Greedy algorithm.

1. Introduction

The evolution and growth of the heterogeneous systems used for high-performance computing have been surprising for many years. Generally, heterogeneous systems [1] for parallel and distributed computing, consist of the system software, multiple processors, memories, and a communication network. Nowadays, more than 60 percent of electronic control unit parts in both automotive electronics and avionics systems employ the heterogeneous distributed embedded system (HDES) [2] because of the benefits in the resource utilization, weight, scale, and power consumption and high-performance. In the HDES architecture, the specialized processing elements are used to accelerate the complex tasks, also coprocessors for general purpose computing cooperated with single or multicore processors are exploited for tasks require flexibilities. There are many types of coprocessors such as application-specific integrated circuit chips, FPGAs (field-programmable gate arrays), GPUs (graphics processing units) architecture. Similar to GPGPUs (general purpose GPUs) produced by Intel Corporation, MIC (many integrated core) is another kind of coprocessor, which is targeted for highly parallel workloads in a large set of scientific data.

Data-dominated applications in a variety of fields, such as computational, engineering, and scientific, are described by a DAG (directed acyclic graph). Nodes represent tasks and edges represent communication messages between tasks in the DAG. The scheduling tasks on HDES processors [3] to minimize the schedule length of a DAG-based parallel scientific application is a well-known nondeterministic polynomial-hard (NP-hard) optimization problem, and numerous meta-heuristic list scheduling algorithms have been proposed to generate near-optimal solutions of the problem. To

implement and orchestrate tasks in HDES, generally, message passing interface (MPI) is employed to make full use of distributed clusters and ensure high scalability. There are many application programming interfaces (APIs) or high-level programming languages used for MPI [3] such as open computing language (OpenCL), open multi-processing (OpenMP), and coprocessing threads (COPRTHR) SDK from Brown Deer Technology.

2. Background and related work

In this paper, there are two study parts conducted for related work consisting of the embedded heterogeneous system (EHS) architectures and meta-heuristic applications on heterogeneous systems. The heterogeneous computing platform (HCP) for data processing in [3] presents the heterogeneous processors and partial reconfigurable hardware accelerators work together with efficiency through APIs consist of COPRTHR SDK and Epiphany SDK on Parallella board. The matrix-vector multiplication in a different size is the benchmark application for HCP, the results show that when data is increasing, the heterogeneous processor and a hardware accelerator are the best performance on the platform.

The work in [5] implements edge detection on heterogeneous system architecture (HSA) environment to compare an ARM multicore processor, Coprocessor and FPGA, and x64 architecture in terms of energy efficiency and optimal performance evaluated from image processing application. The ARM multicore processor with coprocessors (Epiphany multicore processor + FPGA) provides 50 times more energy efficiency better than an x64 architecture in experimental results.

Another interesting use of heterogeneous systems is related to clustering systems. The research in [4] presents a large-scale parallel method of moments on CPU/MIC heterogeneous clusters that proposes hybrid MPI/OpenMP APIs management for distributed CPU/MIC heterogeneous platforms. An implementation of a single board computer cluster for the crypto algorithm is presented in [6] to study 64-board. The cluster architecture is the Raspberry Pi 3 Model B to problem-solving of implementation of the DOZEN crypto algorithm. Towards a dataflow runtime environment for Edge, Fog, and In-Situ computing based on distributed Sucuri architecture [7], also presents the implement on multicore embedded board.

The research work on heuristic algorithms or applications for minimizing the schedule length in heterogeneous computing systems are widely investigated.

The hybrid heuristic-genetic algorithm with adaptive parameters for static task scheduling in a heterogeneous computing system is presented in [8] to further improve the performance of existing algorithms. In [9], a greedy ant algorithm is used for workflow scheduling for heterogeneous computing. This shows a novel workflow scheduling algorithm named Greedy-Ant to minimize the total execution time of an application in heterogeneous environments.

The well-known genetic algorithm is employed for efficient parallel execution on Epiphany manycore processors. The work in [10] shows an evaluate Parallella which is a small board with the Epiphany coprocessors consisting of sixteen MIMD cores connected by a mesh network-on-a-chip. The results are based on classical genetic algorithms.

The scheduling tasks on HDES processors [11] to minimize the schedule length of a DAG-based parallel scientific application is a well-known nondeterministic polynomial-hard (NP-hard) optimization problem, and numerous heuristic list scheduling algorithms have been proposed to generate near-optimal solutions of the problem.

In this paper, the motivation is to implement the meta-heuristic algorithm to workflow scheduling in the cluster-based HDES environment with heterogeneous embedded machines. We conduct the evaluation and implementation of the genetic algorithm (GA), the ant colony optimization algorithm (ACO), the modified greedy genetic algorithm. (MGGA), the modified greedy ant colony optimization algorithm (MGACO), and genetic simulated annealing (GSA) on the proposed HDES system architecture. Rather comparing with conventional algorithms such GA and ACO, the modified heuristic algorithm such GSA are also included. The assumptions of HDES including task execution and communication models based on the architecture are described. The comparisons in terms of the quality of the results, i.e. scheduling time or makespan, are carried out. The contribution of this research can be summarized by the following: the design framework for data-dominated HDES, the evaluation, and implementation of GA-based HDES and the comparison of HDES with related works.

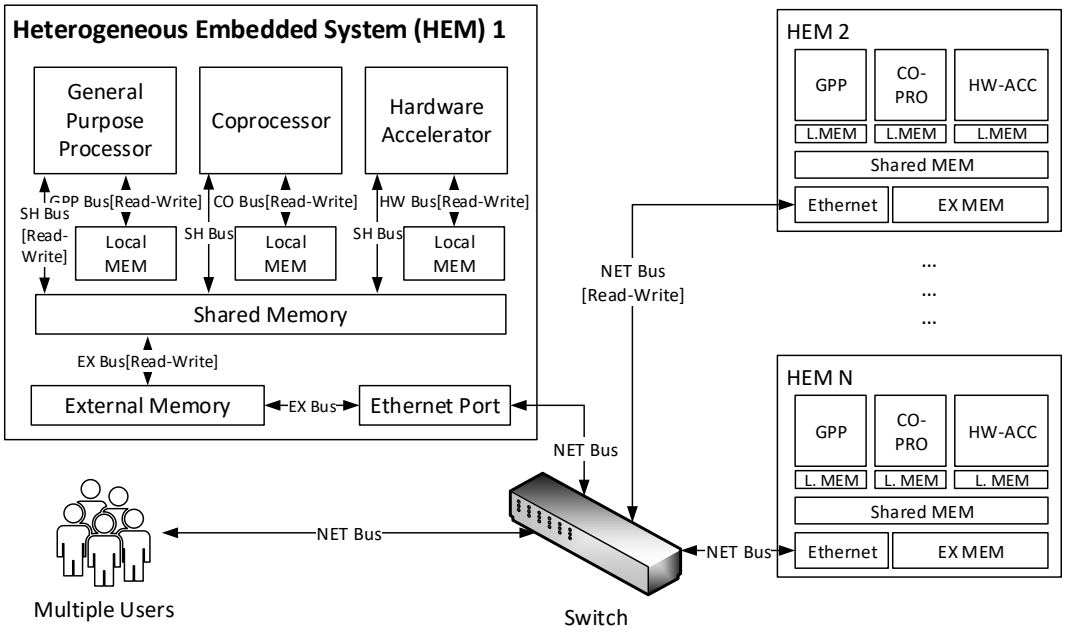


Figure.1 A cluster of heterogeneous embedded machine

The remainder of this paper is organized as follows. Section 3 presents an overview of design and problem descriptions. Section 4, we propose the heuristic algorithm implemented on HDES. Section 5 presents the experimental results and evaluation, respectively. Finally, section 6 concludes the paper.

3. System design overview

This section presents an overview of the system that includes the proposed HDES architecture, design, and heuristic applications that will be implemented and run in the system. HDES architectural design is based on computer architecture principles or embedded system principles [12]. We propose HDES working as a cluster that includes N -heterogeneous embedded machines (HEMs). Each embedded machine contains multiple processing elements such as processors, coprocessors, hardware accelerators also memories, and interconnections. They connect to the network devices for multiple-user services. Rationally, the system performance can be limited by the speed of the network. For this reason, it makes sense to give only parallel processes access to the HDES cluster connection network.

To the best of our knowledge, the optimal arrangement for HDES should be in a mixed model that has attributes of both the asymmetrical and symmetrical design. For an asymmetrical multiprocessor, the back-end processors are used exclusively for executing parallel programs, while asymmetrical multiprocessors, as all processors, have identical functionality running in the same manner.

3.1 HDES architectural design

The multiple users can be accessed to N-layer HDES that includes N heterogeneous embedded machines (HEMs). All are connected and communication through a network device (e.g. router/switch/hub).

Generally, there are 4 layers involved in implementing applications in heterogeneous systems, namely an application layer, an API layer, an operating system (OS) layer, and the hardware layer. The proposed heterogeneous cluster is shown in Fig. 1. In this research, the application layer represents data-dominated applications. The API layer is the set of functions or libraries that are available for the users to manage system resources and run applications. For example, the OpenMP is designed for shared memory, which can run on a HEM cluster using both OpenMP and MPI designed for distributed memory [13] such as local memory, external memory, and bus system. There are many buses used as the communication paths between processing units in this system such as the processor bus (GPP bus), coprocessor bus (CO bus), hardware bus (HW bus), external bus (EX bus), and network bus (NET bus) as shown in Fig. 1. All communication times thru different types of buses are taken into account when scheduling will be shown later in Section 3.4. The OpenCL is an open industry standard for programming a heterogeneous processor, collection [14] of general purpose

processor, coprocessors, and hardware accelerators. It provides efficient accesses to utilize the power of the HEM cluster. Another interesting API is the COPRTHR SDK that provides the libraries and tools [15] for simplifying the use of the HEM cluster for developers targeting coprocessors, and hardware accelerators.

The operating system software layer such as embedded Linux or embedded Windows provides the common services for applications and acts as an interface between API and the hardware. Lastly, the hardware layer is a single board computer that includes heterogeneous processors, buses, memories, and storage. In this paper, we focus on the Parallella board from Adapteva Inc [16] consists of the host processor as Xilinx Zynq Dual-core ARM A9 XC7Z020, the coprocessor is Epiphany 16-core CPU E16G301, 1GB DDR3 memory, gigabit high speed Ethernet, 128MB QSPI Flash, micro-HDMI, 48GPIOs, as the hardware layer.

3.2 Reconfigurable arrays

The flexibility and reusability of a reconfigurable array or an FPGA as a hardware accelerator is due to its configurable logic blocks that are interconnected through configurable routing resources [17]. The reconfigurable arrays are built to support specific tasks or jobs such as digital signal processing, image processing [5], and video processing. In this section, we briefly demonstrate two applications from reconfigurable arrays on the Parallella board that contains a host processor, a Xilinx Zynq, consist of an ARM dual core processor and reconfigurable arrays. A high-speed Epiphany Link (eLink) bus communication is between the host processor and Epiphany 16-core coprocessor.

To utilize reconfigurable arrays, contain 80,000 logic cells and 220 digital signal processing (DSP) slices, IP customization from RTL source files can be written in HDL codes or automatically generated using the Xilinx Vivado design suite as shown in Fig. 2. This IP interface mode can be configured as master or slave, also data width and memory sizes. There are three AXI interfaces for IP customization as AXI4; for memory-mapped interfaces, AXI4-Lite; for single transaction memory-mapped interfaces, and AXI4-Stream; for high-speed streaming data.

3.3 Meta-Heuristic algorithms for HDES partitioning and scheduling problem

There are many meta-heuristic algorithms for searching efficient scheduling that aim for optimal

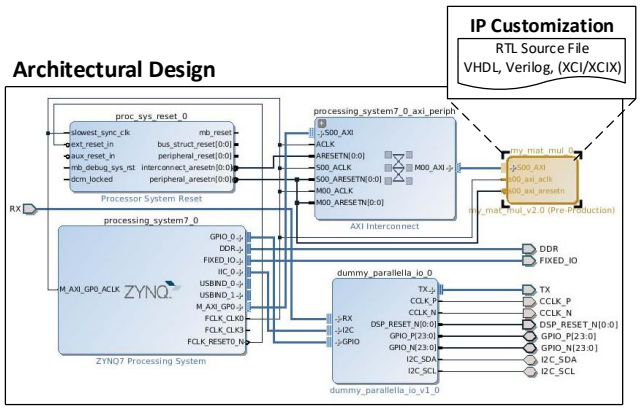
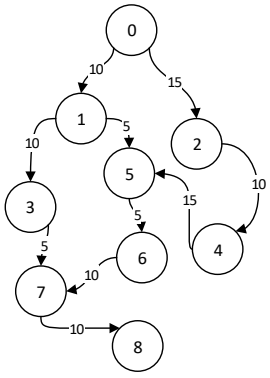


Figure.2 IP customization and architectural design



Execution Time (ns)

Task (**T**)	GPP	CO-PRO	HW-ACC
0	35	20	10
1	40	20	10
2	45	15	25
3	35	40	10
4	40	35	25
5	50	15	40
6	30	35	30
7	30	20	10
8	25	15	30

Figure.3 A sample task DAG. The table shows execution time of task on the processors

task scheduling such as genetic algorithm (GA), simulated annealing (SA), Tabu search (TS), ant colony optimization algorithm (ACO), and particle swarm optimization (PSO). In this paper, we aim to implementation GA and ACO to solve the optimization problem based on HDES.

3.4 Problem description

We introduce a set of execution time model on the right side of Fig. 3 of the heterogeneous system processes consists of GPP, coprocessor, and a hardware accelerator. The minimum execution time is normally from hardware accelerator since it can utilize parallel processing. The maximum execution time is from GPP since it is designed for flexible uses that support different types of jobs.

On the left side of Fig. 3, a data-dominated application with nine tasks. A set of tasks with precedence constraints is modeled in directed acyclic graphs (DAGs). This paper, $G(V, E)$ represents a task DAG, where V is a set of tasks or nodes and E is a set of edges or data communication. Each node $v_i \in V$ indicates a task. Each directed edge $e(i, j) \in E$ represents dependency constraint between v_i and v_j such that v_i should be completed

before v_j can be started, where *i* and *j* define as row and column, respectively.

The nodes of the graph represent the tasks, while the edges denote the precedence of the task. The number on edges represents the amount of data communication ranging from 5, 10, and 15 bytes. Communication time, which is a product of data and bus speed, depends on the types of buses in the system including processor bus (GPP bus), coprocessor bus (CO bus, hardware bus (HW bus), external bus (EX bus), and network bus (NET bus) as mentioned previously.

Fig. 4 shows the example of partitioning results generated by GA in the HDES system contains 3 HEMs. As can be seen, Task 0, 1 and 2 are in HEM1 where Task 0 is running in coprocessor while Task 1 and 2 are running in hardware accelerator.

Fig. 5 shows the scheduling scheme. The scheduling time or makespan obtained from this simulation is 650ns. We can see that communication time where tasks are located in different processing elements and are taken into account. For example, Task 2 (implemented in HEM1 coprocessor) requires SH bus, EX Bus, and NET Bus for sending output data thru the network switch to Task 4 (implemented in HEM2 GPP). Task 7 does not require a network communication bus to send out data to Task 8 since both are implemented in the same HEM.

The $makespan$ represents the actual finish time (AFT) of the exit task v_{exit} and is defined as:

$$makespan = AFT(v_{exit}), \tag{1}$$

where v_{exit} is the summation value of all tasks v_i and v_j.

The objective function defined as $\Im(v_{exit})$ is a goal of an optimization problem to be minimized, that can be formulated as follows.

$$\Im(v_{exit}) = \min_{v_i \epsilon V, e_i \epsilon E} \{AFT(v_i, v_j)\}, \tag{2}$$

$$AFT(v_i, v_j) = \Gamma_{v_i,d_i} + \kappa_{pen}(v_i, d_i), \tag{3}$$

$$\Gamma_{v_i,d_i} = [t_{exe}(v_i, d_i) + t_{com}(d_i) - t_{conc}(v_i, d_i)], \tag{4}$$

where $t_{exe}(v_i, d_i)$ is a set of task execution time in directed acyclic graphs (DAGs), $t_{com}(d_i)$ is the data communication dependencies between tasks, $t_{conc}(v_i, d_i)$ is task concurrency value, and d_i is the i^{th} of the communication time. This $t_{conc}(v_i, d_i)$ equals to the minimum execution time if

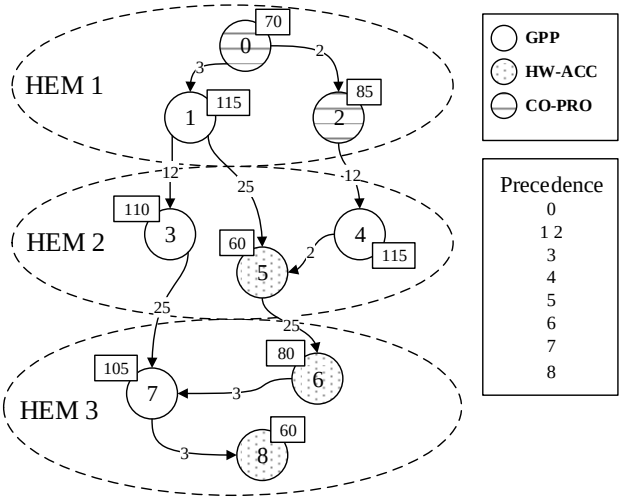


Figure.4 Partitioning result of HDES by GA

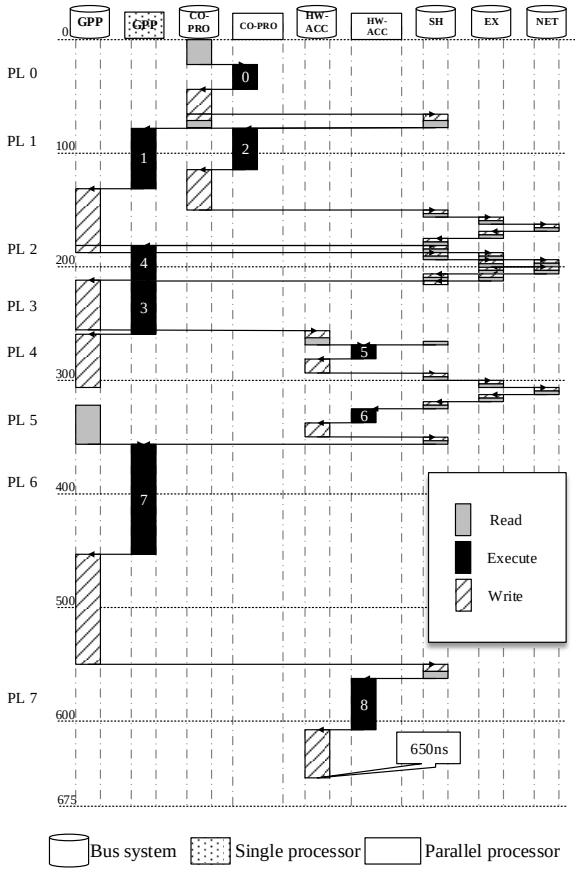


Figure.5 Scheduling result of HDES by GA

there are multiple tasks running in parallel. In a task graph, $e(i,j)$ is associated with a weight which represents the communication time between v_i and v_j when v_i and v_j are assigned to different processing elements. Note that the communication time is negligible if there are two tasks implemented in the same place.

Only the longest execution time will be accumulated into the objective function. $\kappa_{pen}(v_i, d_i)$ is calculated by the constraint values as shown in Algorithm 1. Line 1 shows the create parameters,

N_{cores} is number of cores per a HEM cluster, Per_{logic} is percentage of logic cells per HEM. Line 2-9 show the decision of the constraint values using in the objective function. Note that the Max_{cores} is the maximum number of coprocessor cores in the system, i.e. Epiphany has 16 cores.

Algorithm 1. Constraint

```
1: CREATE the parameters: N_cores, Per_logic
2: if (1 ≤ N_cores ≤ Max_cores ) do
3:    | Constraint <= 0;
4: else if (1 ≤ Per_logic ≤ Max_cores ) do
5:    | Constraint <= 0;
6: else
7:    | Constraint <= 1;
8: end if
9: OUTPUT the constraint
```

In this paper, the $l \times m$ computation matrix MAT stores the execution time of tasks V running on HEM M. $l = |V|$ is the number of tasks and $m = |M|$ represents the number of processing elements in the system. The element $MAT_{i,j}$ is the execution time of task v_i on HEM m_t.

$$MAT = \begin{bmatrix} MAT_{i,j} & MAT_{i,j+1} & \cdots & MAT_{i,j+l} \\ MAT_{i+1,j} & MAT_{i+1,j+2} & \cdots & MAT_{i+1,j+l} \\ \vdots & \vdots & \ddots & \vdots \\ MAT_{i+l,j} & MAT_{i+l,+l} & \cdots & MAT_{i+l,j+l} \end{bmatrix}. \quad (5)$$

The comparisons are performed based on the metrics of schedule length ratio (SLR) and speedup.

The SLR, a key measurement of a scheduling algorithm is the $makespan$ of the schedule obtained. Due to the distinct graph topology, $makespan$ should be normalized to a lower bound. The SLR is defined as:

$$SLR = \frac{makespan}{\sum_{v_i \in CP_{min}} (minMAT_{i,j})}. \quad (6)$$

The summation of the minimum execution time of tasks is on the critical path CP_{min}. For any scheduling algorithms, the SLR value of a graph is larger than one. Therefore, the lower SLR value is the better performance the algorithm will have.

The $Speedup$ is defined as the ratio of the sequential execution time to the parallel execution time (i.e., $makespan$):

$$Speedup = \frac{\sum v_{i \epsilon\, GPP} t_{com}}{makespan}, \quad (7)$$

where $\sum v_{i \epsilon\, GPP} t_{com}$ is the sequential execution time with the data communication dependencies between tasks computed by assigning all tasks to a single machine as GPP.

4. Proposed heuristic algorithms based on the HDES architecture

We introduce four meta-heuristic algorithms for scheduling problem based on the proposed HDES architecture. The algorithms include genetic algorithm (GA), ant colony optimization algorithm (ACO), modified greedy genetic algorithm (MGGA), and modified greedy ant colony optimization algorithm (MGACO).

4.1 GA implementation on HDES

The genetic algorithm to the matching problem requires the setting up some parameters and some adaptation. There are six steps of GA proposed as follows.

At first step, the chromosome encoding represents a sequence of task $t = \{t_0, t_1, \ldots, t_n\}$ of N layers of HEM cluster. Each HEM obtains 3 consecutive genes partly inside the chromosome that represent the mapping results in processing element, i.e. processors (GPP), coprocessor (CO-PRO) and hardware accelerator (HW-ACC). For example, a HEM0 contains the series of tasks $S = \{s_0, s_1, s_2\}$ implemented in GPP, HW-ACC, and CO-PRO respectively. The task s_α can be expressed as:

$$s_\alpha = \alpha + (n+2),\ \alpha = 0,1,2; n = 1,2,\ldots,N; \quad (8)$$

where α denotes as a type of processors.

The second step, an initial population is generated randomly as a sequence of integers inside a chromosome representing the class indices in one of the matched DAG.

The third step, an evaluation that each candidate solution represents a sequence of tasks in the DAG. This sequence is given as GA to find the corresponding injective match. The sum of the similarities between the matched elements in this injective match is used as the objective function, the higher summation is the fitter result.

The fourth step, the selection is the procedure by which R, where $R < N$ individuals that are chosen from the population of HDES for reproduction. We employ a roulette wheel selection to select parents from the population. Each chromosome k belongs to the parent population with the probability of P_k. The roulette wheel is created by calculating a cumulative probability for all chromosomes as follows:

$$Q_i = \sum_{k=1}^{i} P_k,\ i = 0,1,2,\ldots,N; \quad (9)$$

where Q_i is a cumulative probability of the i^{th} solution.

The selection of R parents is accomplished by spinning the wheel R times. Each spin is tantamount to a generated random number in the range [0,1].

In the fifth step, the crossover occurs between the two parents. We use a simple technique, random one-point crossover. A random rate is 25 percent. The crossover operator builds an offspring by random choosing to cut-points in the two parents, copying the subsequences between the cut points in the two parents into new two offspring, one each, and then the remaining indices are filled, position wise, from the other parent.

The one-point crossover is the generation of two children from two N -dimensional parent $Parent1\ (\beta_0, \beta_1, \dots, \beta_N)$ and $Parent2\ (\alpha_0, \alpha_1, \dots, \alpha_N)$, where β_N and α_N are individual task N. Let ξ denote the crossover point. Therefore, the offspring of parents are generated as follows:

$$Child_0^{new} = (\beta_0, \beta_1, \dots, \beta_\xi, \alpha_{\xi+1}, \alpha_{\xi+2}, \dots, \alpha_N), \quad (10)$$

$$Child_1^{new} = (\alpha_0, \alpha_1, \dots, \alpha_\xi, \beta_{\xi+1}, \beta_{\xi+2}, \dots, \beta_N). \quad (11)$$

The final step, the mutation is performed by a random selection of a position in the chromosome and swapping its task probabilistically with a nonuniform mutation technique from the pool of chromosomes. Nonuniform mutation induces an increasingly localized search for optimal solutions in which the sets of genes that are chosen for mutation are defined utilizing boundaries.

Let $Soln\ (\gamma_0, \gamma_1, \dots, \gamma_i, \dots, \gamma_N)$ be a system solution of an optimization problem and its i decision variable (γ_i) be selected for mutation. Nonuniform mutation produces a mutated solution $Soln\ (\delta_0, \delta_1, \dots, \delta_i, \dots, \delta_N)$, whereby δ_i is calculated as follows:

$$\delta_i = random(\gamma_i - \hbar, \gamma_i + \hbar), \quad (12)$$

$$\hbar = \hbar_0 \times \frac{iteration_{max} - iteration_{now}}{iteration_{max}}, \quad (13)$$

where $\hbar_0$ is an initial value of $\hbar$, $iteration_{now}$ defines as current iteration, and $iteration_{max}$ is a maximum iteration.

4.2 MGGA implementation on HDES

The modified greedy genetic algorithm (MGGA) is shown on the left side of Fig. 6, to a matching problem in DAGs. As it can be seen that MGGA is similar to the GA, but one greedy step is added on. The extra step of MGGA called Greedy GA as shown in Algorithm 2. The input of this algorithm is a size

Algorithm 2. Greedy GA

```
1:  CREATE the parameters: pop_size, pop_next, con
2:  INITIAL con_greedy
3:  for (j = 0; j < (pop_size - 1); j++)
4:      | for (i = 0; i < (popsize - 1); i++)
5:      |   | if (pop_next[i + 1]. con <
            pop_next[i]. con_greedy) then
6:      |   |   | select the best constraint of pop_next
7:      |   | end if
8:      | end
9:  end
10: OUTPUT the pop_next is the best solution
```

of the population with constraint. Line 1 shows creating parameters as pop_{size} is population size, pop_{next} is next population and con is a constraint. Line 2 shows initial con_{greedy} as greedy constraints, which is set to zero. Line 3 shows loop process of row of population size subtracted by one. Line 4 shows for the loop process of column of population size subtracted by one. Line 5-7 show the decision instead of the solutions to find the best greedy constraint for population next. And Line 10 shows the best solution of MGGA output.

4.3 ACO implementation on HDES

To employ the ant colony optimization algorithm, we require the setting up some parameters and some adaptation.

The first step, initial parameters includes the decision variable values $(\varepsilon_1, \varepsilon_2, \varepsilon_3, \dots, \varepsilon_N)$ which are chosen from a set of predefined values. Each decision variable i takes a value from a predefined set of values E_i as follows:

$$E_i = \{\varepsilon_{i,1}, \varepsilon_{i,2}, .., \varepsilon_{i,\zeta}, \dots, \varepsilon_{i,Z_i}\}, i = 1,2, \dots, N; \quad (14)$$

In which set of E_i predefined values for the decision variable, $\varepsilon_{i,\zeta}$ is a possible value for the decision variable, and Z_i is a total number of possible values for the decision variable.

The second step, to generate a set of ants as an array of $1 \times N$ that describes the ant's path. This array is defined as:

$$Ant = \{\eta_1, \eta_2, \eta_3, \dots, \eta_i, \dots, \eta_N\}, \quad (15)$$

where Ant is a solution of the optimization problem, η_i the decision variable of solution Ant, and N is the number of decision variables.

The third step, to allocate pheromone into objective function, the pheromone is a set of values that make better solutions achieve higher concentration of pheromone in comparison with

values that make worse solutions. N arrays of size $1 \times H_i$ are employed to allocate pheromone to the decision space so that each of them is assigned to one decision variable as follows:

$$\Phi_i = \{\phi_{i,1}, \phi_{i,2}, .., \phi_{i,h}, \dots, \phi_{i,H_i}\}, \quad (16)$$

where $i = 1,2, \dots, N$.

In which Φ_i is pheromone matrix for the decision variable and $\phi_{i,h}$ is pheromone concentration of the h possible value of the i decision variable. The elements of the matrix Φ_i are equal to zero at the beginning of the algorithm optimization. The pheromone allocation is achieved by increasing the pheromone levels associated with a chosen set of good solutions. The concentration of pheromone for the possible value of the decision variable is updated as:

$$\phi_{i,h}^{new} = (1-\rho) \times \phi_{i,h} + \sum_{j=1}^{n} \Delta\phi_{i,h}(j), \quad (17)$$

where $\phi_{i,h}^{new}$ is new concentration of pheromone of the possible value of the decision variable, ρ is an evaporation rate, and $\Delta\phi_{i,h}(j)$ defines as the quantity of pheromone laid on the h possible value of the i^{th} decision variable by the j^{th} ant.

The fourth step, a generation of new ants for each decision variable i is assigned a value with a probability that depends on the concentration of pheromone. A cumulative probability for all the possible values of each decision variable as follows:

$$\Psi_{i,g} = \sum_{q=1}^{g} P_{i,q}, \quad (18)$$

where $i = 0,1,2, \dots, N$ and $g = 1,2, \dots, \Theta_i$.

In which $\Psi_{i,g}$ defines as cumulative probability of the g possible value of the i^{th} decision variable.

4.4 MGACO implementation on HDES

The modified greedy ant colony optimization (MGACO) algorithm is shown on the right side of Fig. 6 with one greedy step adding. The step is called Greedy ACO in Algorithm 3. The input parameters of this algorithm are the path of ant named $length$ with a constraint, levels, and route of the pheromone. Line 1 shows creating parameters as $length_{best}$ is the optimal solution, $length$ is a good solution, con is a constraint, $route_{best}$ is the route of increasing pheromone levels. Line 2 shows initial $length_{best}$ greedy constraints of optimization solutions. Line 3-8 show the decision instead of the solutions to find the optimal solution. Line 5 shows the loop process of replacing pheromone levels. And Line 9 shows the MGACO output.

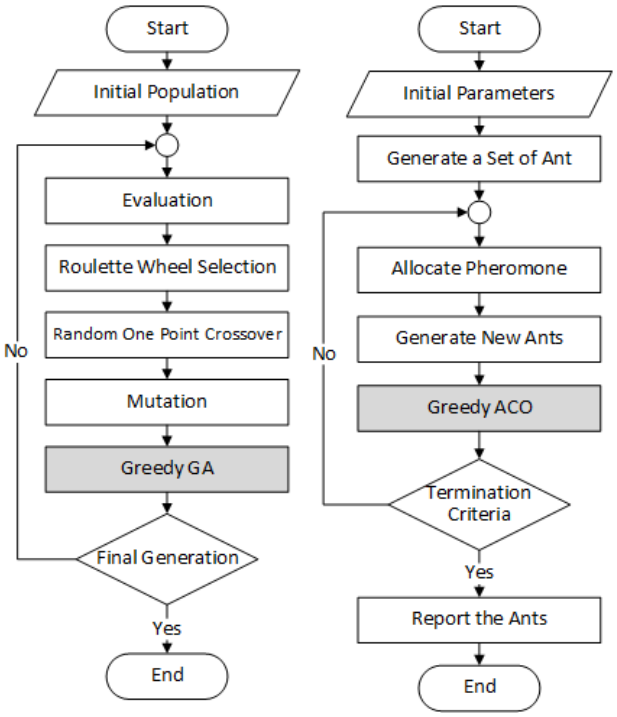


Figure.6 The flow chart of proposed MGGA (left) and MGACO (right)

Algorithm 3. Greedy ACO

1: CREATE the parameters $length_{best}$, $length$, con, and $\text{route}_{\text{best}}$
2: INITIAL $length_{best}$
3: **if** ($length < length_{best}$) **then**
4: | $length_{best} = length.con$;
5: | **for** (i = 0; i < number of tasks; i++)
6: | | $route_{best}[i] = route.con[i]$;
7: | **end**
8: **end if**
9: OUTPUT the best of solution

5. Experiments and results

In this section, experiments are carried out with the comparison of the results using the heuristic algorithms to find optimal solutions for HDES. The experimental design of HDES that includes the 9-layer of HEM connected by a network switch based on a star topology. The algorithms with different task graphs are simulated in this heterogeneous embedded system.

5.1 HDES hardware setup

The HDES hardware environment as shown in Fig. 7 consists of nine the Parallella boards, known as HEM, and a 16-port of D-Link DGS-1016D

Figure.7 The HDES hardware setup

gigabit switch connects with CAT5e LAN cable. The users can monitor on the laptop through wireless LAN. The operating system is the Parabuntu 2016.11.1, embedded Linux, official Ubuntu distro for Parallella installed that is open source from the community of www.parallella.org. The APIs are operated using a collection of COPRTHR SDK from Brown Deer Technology with the Epiphany SDK from Adapteva.

5.2 Parameters and random task graphs configuration

For GA and MGGA, the chromosome encoding, represents a sequence of task $t = \{t_0, t_1, \ldots, t_n\}$ of N layers of HEM cluster as described in Section 4. The roulette wheel is employed as a selection technique. For ACO and MGACO, there are 50 ants. The pheromone parameter Φ_i is 1.

The DAGs random task graphs are randomly generated. The parameters for DAG generating software are set as below.

- The number of task nodes in a DAG is {10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, 500}.
- Communication-to-computation ratio (CCR) is the ratio of average communication cost to the average computation cost as {0.1, 0.2, 0.5, 0.8, 1, 2, 5, 8, 10} [9].
- Range of computation costs on machines: the heterogeneity complex factor for the machine performance on speed is {0.1, 0.2, 0.5, 1, 2} [9].
- The number of levels spanned by communications is {1, 2, 4}.
- The similarity of task numbers between {0.2, 0.5, 0.8} which a high value indicates the higher similarity.
- The parallelism degree of a DAG is {0.1, 0.4, 0.8} which is a shorter DAG with high parallelism.
- The dependency degree of the nodes in a graph is {0.2, 0.5, 0.8} which the high dependency is larger density.

All algorithms are written in C/C++, complied by GCC/G++ on HEMs inside the HDES. For each problem instance, the algorithm will run for 3000 iterations, which is our stopping criterion, and the most feasible solution is then acquired.

5.3 The Results and Discussion

With MPI, the meta-heuristic algorithms can be managed to run on selected HEM using master-slave technique. The results from different sizes of task graphs in this heterogeneous embedded system types of algorithms are collected at the HEM master. The best solutions are averaged and compared using makespan, speedup, and SLR metrics.

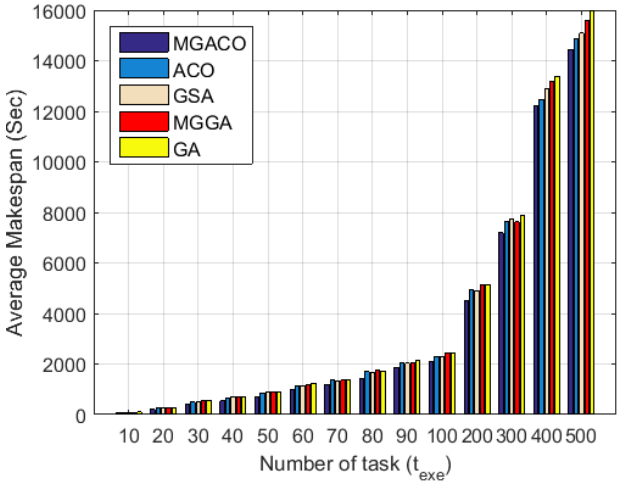


Figure.8 The average makespan compared with different algorithms on random task graphs.

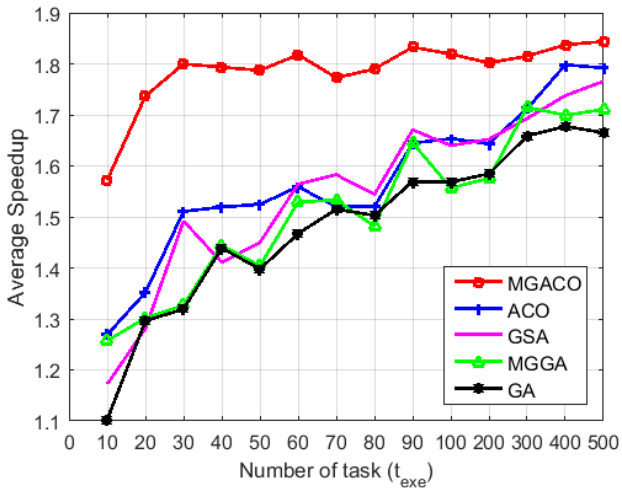


Figure.9 The average speedup compared with different algorithms on random task graphs.

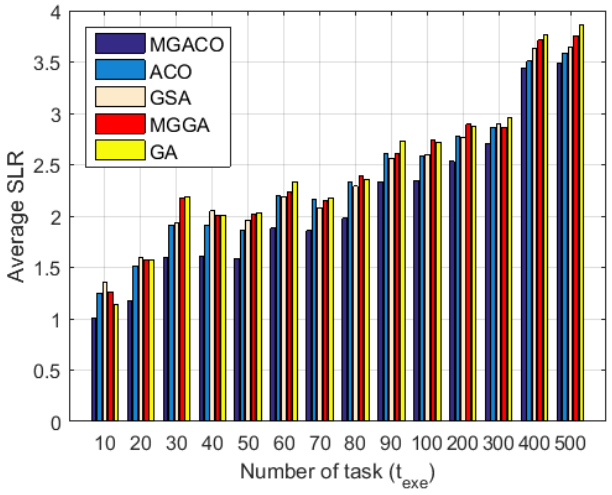


Figure.10 The average SLR compared with different algorithms on random task graphs

Fig. 8, shows the average makespan from all DAG graph sizes obtained from GA, ACO, MGGA, MGACO, and genetic simulated annealing (GSA). The GSA algorithm [18] utilizes the simulated annealing (SA) algorithm in the part of mutation operation based on GA for finding the optimization value. The average makespan of MGACO is 240 minutes which is the best of overall average makespan while ACO is 247 minutes, GSA is 251 minutes, MGGA is 259 minutes, and GA is 266 minutes, respectively.

In Fig. 9, the proposed MGACO performs better than other algorithms. Compared with GA and GSA, MGACO improves speedup by 30 percent for a small graph size. When the task number increases, MGACO shows dramatically in advantage. It is found that the performance of MGACO tends to converge rapidly at the steady state, when the number of tasks is larger than 30. Although, ACO performs better than GSA, MGGA, and GA, when the task number exceeds 50, its performance decreases afterwards. When number of tasks is equal to 80, MGACO improves speedup about 33 percent.

For the SLR, Fig. 10 shows the details of the comparisons on random tasks graphs. It is obvious that MGACO performs better than the other algorithms. However, when n = 60, 70, 80 due to the lower boundary of SLR has been reached, all the algorithms are roughly equal. In term of speed up and SLR, the overall results reveal that MGACO outperforms ACO, GSA, MGGA and GA to find optimal task scheduling in the proposed HDES architecture.

All algorithms in the experiments simulate an initial task sequence generated randomly. GA, GSA, and MGGA are based on the genetic algorithms, while MGACO and ACO are list-based on the ant colony method. Figs. 8, 9, and 10 clearly demonstrate that MGACO outperforms ACO, GSA, MGGA, and GA concerning the metrics, including makespan, Speedup, and SLR.

As can be seen, the performance of the algorithm searching for more quality solution can be increased by using a simple greedy algorithm inserted into a selection stage of conventional heuristic algorithms. It usually finds a near optimal solution in polynomial time. Theoretically, the performance analysis of using greedy algorithm to solve basic optimization problems of acyclic graphs is elaborated in [19]. The ratio of a feasible solution with an optimal solution is never more than twice for most problem instances.

By introducing a simple greedy algorithm in meta-heuristic searching for a shortest makespan in this paper, MGACO can generate better scheduling results compared to GA and MGGA because an ant colony can help to find the best pathway. As the complexity of the task graph grows, the task graph becomes harder for them to produce consistent results on a variety of graphs. The resource constraints are addressed in the cost function since the number of reconfigurable cells and coprocessors are limited.

6. Conclusion

This paper considers the genetic algorithm and the ant colony optimization modified with the greedy algorithm and presents an approach for ACO based workflow scheduling. A new meta-heuristic information based on forward dependency is proposed to build probability for ACO to generate task priorities. Additionally, a greedy algorithm for machine allocation is incorporated to complete scheduling. Based on the random task graphs, experimental results in the HEM cluster demonstrate the effectiveness of the modification of greedy ant colony optimization algorithm which outperforms the others by 33% more result quality. Future work would be a real implementation of data-dominated applications in this HDES based on the optimal scheduling results.

References

[1] M. Zahran, *Heterogeneous Computing: Hardware and Software Perspectives*, Vol.1, ACM Books, New York, N.Y.2019.

[2] J. Huang, R. Li, J. An, D. Ntalasha, F. Yang and K. Li, "Energy-Efficient Resource Utilization for Heterogeneous Embedded Computing Systems", *IEEE Trans. on Computers*, Vol.66, No.9, pp.1518-1531, 2017.

[3] S. Prongnuch and T. Wiangtong, "Heterogeneous Computing Platform for data processing", In: *Proc. ISPACS*, pp.1-4, 2016.
[4] Y. Chen, S. Zuo, Y. Zhang, X. Zhao, and H. Zhang, "Large-Scale Parallel Method of Moments on CPU/MIC Heterogeneous Clusters", *IEEE Trans. on Antennas and Propagation*, Vol.65, No.7, pp.3782-3787, 2017.
[5] S. Prongnuch and T. Wiangtong, "The Implementation of Edge Detection on HSA Environment", In: *Proc. iEECON*, pp.1-4, 2017.
[6] A. B. Vavrenyuk, A. S. Rusakova, A. A. Radiskhlebova, M. A. Ivanov, and V. V. Makarov, "Implementation of the DOZEN Cryptoalgorithm on the Cluster of Single-board Computers", In: *Proc. ELConRus.*, pp.369-372, 2019.
[7] S. N. Agathos and V. V. Dimakopoulos, "Adaptive OpenMP Runtime System for Embedded Multicores", In: *Proc. BUC*, pp.174-181, 2018.
[8] S. Ding, J. Wu, G. Xie, and G. Zeng, "A Hybrid Heuristic-Genetic Algorithm with Adaptive Parameters for Static Task Scheduling in Heterogeneous Computing System", In: *Proc. IEEE Trustcom/ BigDataSE/ICESS*, pp.761-766, 2017.
[9] B. Xiang, B. Zhang, and L. Zhang, "Greedy-Ant: Ant Colony System-Inspired Workflow Scheduling for Heterogeneous Computing", *IEEE Access*, Vol. 5, pp.11404-11412, 2017.
[10] L. Faber and K. Boryczko, "Efficient Parallel Execution of Genetic Algorithms on Epiphany Manycore Processor", In: *Proc. FedCSIS*, pp.865-872, 2016.
[11] G. Xie, G. Zeng, X. Xiao, R. Li and K. Li, "Energy-Efficient Scheduling Algorithms for Real-Time Parallel Applications on Heterogeneous Distributed Embedded Systems", *IEEE Trans. on Parallel and Distributed Systems*, Vol.28, No.12, pp.3426-3442, 2017.
[12] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Vol.6, Morgan Kaufmann, San Francisco, C.A.2017.
[13] P. Czarnul, *Parallel Programming for Modern High Performance Computing Systems*, Vol.1, Chapman and Hall/CRC, New York, N.Y.2018.
[14] L. Howes, *The OpenCL Specification, Version: 2.1, Khronos OpenCL Working Group*, [Online] Available: https://www.khronos.org/registry/ OpenCL/specs/opencl-2.1.pdf. Accessed on: Sep. 2, 2019.
[15] Brown Deer Technology, *COPRTHR-2 Overview*, [Online]. Available: http://www. browndeertechnology.com/docs/COPRTHR-2 Overview rev-b-20160629.pdf. Accessed on: Sep. 2, 2019.
[16] Adapteva, *Parallella-1.x Reference Manual*, [Online]. Available: https://www.parallella.org /docs/parallella manual.pdf. Accessed on: Sep.2, 2019.
[17] O. Terzo, K. Djemame, A. Scionti, and C. Pezuela, *Heterogeneous Computing Architectures: Challenges and Vision*, Vol.1, CRC Press, Florida, F.L.2019.
[18] M. Zhao, X. Yin, and H. Yue, "Genetic Simulated Annealing-Based Kernel Vector Quantization Algorithm", *International Journal of Pattern Recognition and Artificial Intelligence*, Vol.31, No.5, pp. 1-28, 2017.
[19] L. C. Thanh, "Performance Analysis of Greedy Algorithms for Max-IS and Min-Maxl-Match", *Vietnam Journal of Mathematics*, Vol.36, No.3, pp.327-336, 2008.