Scheduling Problems For Parallel And Distributed Systems

Olga Rusanova and Alexandr Korochkin
National Technical University of Ukraine – "Kiev Polytechnical Institute"
Prospect Peremogy 37, 252056,
Kiev, Ukraine
e-mail: {rusanova, cora}@comsys.ntu-kpi.kiev.ua

1. ABSTRACT

A two-pass scheduling algorithm for parallel and distributed computer systems is presented in this paper. We consider this algorithm as a complex of two stages: process queue formation and assignment procedure. A new approach of both stages realization is proposed. Our algorithm can be used to increase efficiency of static and dynamic scheduling.

1.1 Keywords

Scheduling, parallel, distributed, algorithms, computer systems

2. INTRODUCTION

Today is the decade of development of parallel and distributed systems. They are very perspective in getting high performance. Scheduling is one of the main factors their effective using. Therefore this problem has received considerable attention in recent years. Improvement of scheduling approaches is very real.

There are two main evaluations of scheduling: scheduling performance and scheduling efficiency. Scheduling performance is a minimal total completion time of a parallel program. Efficiency is a time complexity of the scheduling. These evaluations contradict each other. The importance of each depends on the approach used, either static or dynamic. Time complexity is a determining measure for dynamic methods which execute the schedules during the runtime of parallel programs. That's why in this case it is hard to get high scheduling performance. Time complexity is not so significant for static scheduling because this methods execute at compile time.

Thus static scheduling includes more sophisticated analysis and high performance can be obtained. This paper is devoted to static scheduling technique.

SIGAda'99 10/99 Redondo Beach, CA, USA

3. CLASSIFICATION OF STATIC SCHEDULING APPROACHES

We consider classification of static scheduling algorithms (Appendix A). Scheduling is known as NP-complete problem in general. There are only three cases of optimal scheduling solutions which can be used for very small problem size:

- scheduling tree structured graph on n processor system [4];
- scheduling arbitrary graph on 2 processor system [1];
- scheduling "interval order" graph on n processor system [5].

Thus, nowadays, researches have been concentrated of heuristic suboptimal scheduling algorithms.

Today all heuristic approaches can be divided into the three groups: genetic; clustering and list.

Genetic algorithms are applied to relatively easy problems. When applied to problems with very large search space and where the ratio of the number of feasible solutions to the number of infeasible solutions is low, the genetic algorithm will perform no better than a random search. For scheduling of distributed systems and parallel systems with distributed memory it is necessary to take into account relatively tasks and system topology. It's lead to reduce of satisfactory search space decisions. Thus we think that such approach is not the best suitable for this computer systems.

There are two approaches to clustering. In the beginning of the first of them, the original computation (consisting of many tasks) is assumed to be allocated to unlimited number of completely connected processors and then in the followed steps the clusters of tasks are merged and scheduled on the bounded number of processors.

In the beginning of the second approach, the original computation is assumed to be allocated to one processor for

sequential computing. When more processors are available, some tasks are swapped out of the initial processor to other processors to do the computation in parallel and thus to reduce the computing time of the whole computation.

This approaches can be used only as a pre-processing procedure. In the case of large number of tasks and few processors this algorithms tend to produce a cluster number which is much larger than the number of available processors. In such circumstances, clustering procedure doesn't help much to reach the final solution and thus poor performance may result. Moreover clustering algorithms doesn't consider system topology , therefore it can be used only for completely connected systems.

List scheduling technique is in following. Firstly we form the order of tasks according to their priorities. There are many approaches to determine priorities. Then we assign tasks to processors. Assignment procedure can be performed with or without consideration systems topology and communication cost. The best results give the scheduling heuristics in which such parameters as system topology and communication time between relative tasks are considered. So, we suggest that list scheduling is the best suitable for distributed and parallel systems.

4. PROBLEM DEFINITION

Our main goal is to schedule a parallel programs on the parallel (consists of homogeneous processors) or on the distributed (consists of heterogeneous processors) system such that the total completion time of execution the program is minimal. So the maximal measure of performance is an optimization criterion of the static scheduling.

Parallel program can be represented by a directed acyclic graph (DAG) called task graph. This graph has a set of task nodes and a set of directed communication edges. Each node has computation cost and each edge has communication cost. A task receives all input datas before starting execution, it executes until its completion without interruption. The task graph is assumed to be static, which means it remains unchanged during execution.

Processors of parallel or distributed system are connected in a given topology. Each processor has one or multiple (four or six) links for communications. The number of links is input data for scheduling too. The parallel system topology can be represented by an undirected unweighted graph called processor graph. It has a set of processor nodes and a set of communication edges. The distributed system topology can be represented by an undirected weighted graph. Weight of node corresponds to processor performance and weight of edge to channel capacity. Systems with hardwired connections can be represented by real processor graph in which number of edges equals total number of links of this system. Reconfigurable system can be represented by virtual processor graph where number of edges larger than total quantity of links. Virtual processor graph means that we have a set of real graph processors for scheduling.

Processors

time	P1	P2	P3	P4
1	1	2		
2	1	2-5		
3	1	2-5		
4	4	2-5		
5	5			3
6	5			3-6
7	5			
8	5			
9	6			
10	6	7		

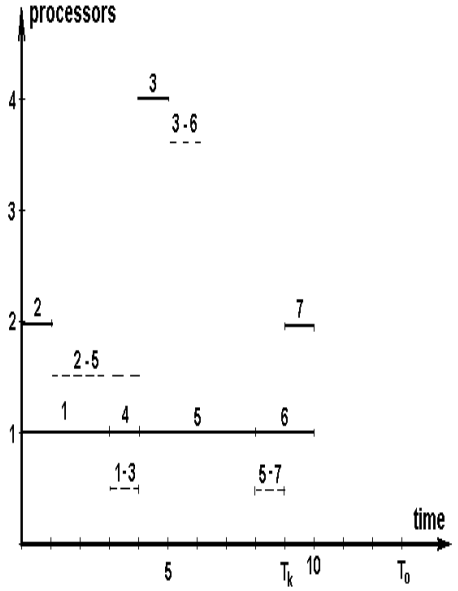


Figure 1. Gantt chart with data transfer

A schedule of the task graph on a processor graph can be illustrated as a Gantt chart, where the start and finish times for all tasks can be easily shown. A Gantt chart consists of a list of all processors and, for each processor, a list of all tasks allocated to that processor ordered by their execution time, including task start and finish times. But an ordinary Gantt chart doesn't show the route of data transfer from one processor to another. We propose to modify the Gantt chart where a condition for each processor includes a list of all allocated tasks and a list of data

transfer all the links. You will see an example of the schedule with data transfer on the picture (fig.1).

There are two key components that contribute to the total completion time: execution time and communication delay. During scheduling we try to minimize both of that components.

As was noted earlier all list scheduling algorithms include two stages:

- order formation of tasks;
- allocation of tasks to processors.

The goal of the first stage is to minimize execution time and the goal of the second stage - to minimize communication delay. To minimize communication delay during allocation procedure we will call the main measures:

- type of communication cost model;
- algorithm choosing processor for allocation;
- minimization delay between the time when some data is produced at some processor and the time when this data can be used by another.

Today there are three main communication cost model.

model-A: in this model each processor can't execute a task and communicate with another processor at the same time. Communication cost equals sum of edges weights in a given task graph;

model-B: this model is similar to model-A but it uses a more practical definition of communication cost. If the task has several succesors and some of them allocated to the same processor, their communication cost is counted only once (In model-A it is counted multiple times);

model-C: this model assumes the existence of an I/O processor. It implies that a processor can execute a task and communicate with another processor (or processors if it has multiple links) at the same time. Communication delay between tasks allocated to the same processor is negligible. Communication delay between two communicating tasks allocated to the two different processors is a function of the size of the message, the route, and the communication speed.

We propose to add the fourth communication model-D. In this model use package commutation. Each message can be divided unto the several units. So, for transit connections we can use pipeline data transmission. We share the viewpoint that for systems with network connections model-D is the best suitable. Otherwise, model-C is the best suitable.

To choose the processor for allocation we propose to take into account topology characteristics of system such as diameter, average diameter and coherence of processors

There are two key components that contribute to the total completion time: execution time and communication delay. During scheduling we try to minimize both of that components.

5. MAIN STAGES ON NEW LIST SCHEDULING HEURISTIC

Firstly we consider the stage of order formation. In list scheduling tasks are considered, for determination order, according to their priorities which guarantee the earliest start and finish time of each task. All known approaches use for that single or multiple characteristics of task graph (such as: slack; finish time; longest to a bottom and to a top nodes; critical path; set of in-neighbors and out-neighbors etc). In this case both stages of scheduling performs sequencially. Otherwise these algorithms don't take into account real delays under the allocation of privious tasks. These delays are connected with bounded number of processors and data communication time.

We suggest that progress of scheduling is in complex implementation of its both interacting stages (order formation and allocation). Our approach is based on this idea. To determine priority of the task we will consider following characteristics: node weight; the earliest and the latest execute time; real slack; sum weight of entering edges. Node weight, the latest execute time and sum weight of entering edges are the characteristics of task graph. Traditionally the earliest execute time is a characteristic of task graph too. But we determine its real value. It depends on real finish time of the preceding tasks after procedure of their allocation. Real slack value is determined by the quantities of the latest and real earliest execute time. It can be used for data transfers. Negative value of the real slack means the execution delay of a given task. We propose to difine the priority for each task as a difference between sum weight of entering edges and the real slack.

Now we consider an allocation procedure.

As was noted earlier to minimize the communication delay we propose to use communication cost models D or C.

Let's consider the procedure choosing processor for allocation. We think that the best of known approaches is the following. Each task assigns to the release processor, which has the minimal path to the processor with the initial data of a given task. If the initial data held at the several processes, the task allocates to the release processor, which has minimal sum of lengths to processors with initial data.

We suggest that the progress of allocation is the following. In our approach a task assigns to the processor with the minimal measure of initial execution time. It doesn't matter if this processor is busy or not at the allocation time.

For parallel system firstly the most urgent task is defined.

If this task is initial in the task graph we allocate it to the free processor with maximal priority. Priorities of

processors are based on the value of average distance to others or on the value of coherence.

For tasks with single predecessor assignment fulfils in a such way. Processor set is devided into the groups with the same distances *(r)* to the processor with initial data of the given task. The search of allocated processor is based on analysis of processor groups with various distances from minimal up to maximal. For each processor group we choose processor with the minimal releasing time. Then we determine start delay ($O_{m,i}$) with following formula :

$$O_{m,i} = \max\{T_m, T_f + e_i * r\} - T_{p,i}$$

where T_m - *m*-processor's release time;

T_f - data formation time for *i*-task;

e_i - entering edge weight of *i*-task;

$T_{p,}$ - latest execute time of i-task.

If the value of start delay isn't positive the assignment performs to this processor, else we go to next processor group. If this value is positive for all processor groups, processor with minimal start delay chooses for allocation.

For tasks with several predecessors assignment is performed in a same way but in this case the start delay is determined with the following formula:

$$O_{m,i} = \max\{T_m, \sum_{j=1}^{n}(T_f + e_{j,i})\} - T_{p,i}$$

where *n* is a number of entering edges and corresponding tasks which forms data for *i*-task.

For distributed systems allocation procedure starts from definition the most urgent task too. Then for initial tasks we allocate to the free processors with maximal performance.

For tasks with single predecessor assignment fulfils in a such way. Processor set we divide into the subsets with the same processor performance. Then each subset is divided into the groups with the same distances to the processor with initial data of the given task. The search of allocated processor is based on analysis of processor subsets from minimal up to maximal processor performance and their groups with various distances (from minimal up to maximal). For each processor group we choose processor with the minimal releasing time. Then we determine the final time execution *($F_{m,i}$)* as follows

$$F_{m,i} = \max\{T_m, T_f + e_i * r\} - T_{p,i} + \lceil w_i / p_m \rceil$$

where w_i - weight of *i*-node;

p_m - *m*-processor performance.

The assignment performs to processor with minimal value of $F_{m,i.}$

For tasks with several predecessors assignment is performed in a same way but the final time execution is determined with the formula

$$F_{m,i} = \max\{T_m, \sum_{j=1}^{n}(T_f + e_{j,i})\} - T_{p,i} + \lceil w_i / p_m \rceil$$

As was noted earlier, communication delay also depends on the difference between the time when some data are produced at some processor and the time when this data can be used by another. To minimize value of this difference, we propose using the second pass of the algorithm. In order to get minimal communication delay we consider a schedule from its begining to the finish and find the cases in which the initial time is larger than their time production. In these situations, we can execute transfers earlier at once after the data production. So, we change our schedule otherwise; we "compress" it and minimize the total completion time of the parallel program.

6. EXPERIMENTAL RESULTS

We consider experimental results of our approach.

We designed a program modeling our algorithm on a PC using the Ada 95 language. It can be used for computer systems of any topologies with the number of processors (nodes) up to 256. Each processor has one or multiple links.

In the first experiment we tested performance aspects of our algorithm. To determine the performance measure, we can use a value as the ratio of the average execution time on the system with random topology after our algorithm to the average critical path length of the random DAGs. (It is the absolute minimum computing time the DAGs need any computing system with any processor numbers.)

As the initial data for this experiment, we used sets of DAGs and system topologies generated randomly. We divided random DAGs into the five groups. Each of them is distinguished by value of the grain size as a ratio of an average execution cost to an average communicatiom cost. Each of these group we test on the ten sets of random system topologies. Each set is distinguished by value as a ratio of an average number of DAGs nodes to an average number of processors.

The results of this experiment are shown in Appendix B in Table 1. Five groups of task graphs form rows and ten sets of system topologies form columns. From the results, we can see that performance improves when the grain size of DAGs increases and the ratio dimensions of the task graph and computing system decreases. In Table 1 :

A - a ratio of an average execution cost to an average communication cost (grain size); B - a ratio of an average number of DAGs nodes to an average number of processors.

Performance measure is a value as a ratio of an average execution time to an average critical path length of the random DAGs.

In the second experiment we compared the performance measure of our algorithm with two algorithms called Mapping, and Duplication with Mapping. These algorithms were designed at the University of Nebraska (Omaha) and can be used for distributed memory systems with any topologies [2,3]. In this experiment we applied the random DAGs with a ratio of execution time to communication cost as 70/30. We tested algorithms on the set of system topologies as follows:

- completely connected;
- ring;
- star;
- mesh with toroid connections;
- tree;
- hypercube.

We tested the mesh topology with number processors equal to 9,16,25,36; tree topology with the number processors equal to 7,15,31,63 and the rest topologies with number processors equal to 8,16,32,64. The results of the second experiment are shown in Appendix B in Tables 2-7, where N – number of processors; MA – "Mapping Algorithm"(El-Rewini, T.Lewis); MDA – "Mapping with Duplication Algorithm (El-Rewini,H.Ali); HMSA – "Heuristic Mapping and Scheduling Algorithm"(proposed).

From the results we can see that our proposed algorithm has the best average performance measure.

Our approach can be used to solve the following tasks:

- system configuration tuning;
- automatic creation of file configuration as the part of parallel program;
- search of the chips for systems with a given topologies;
- design of scalable systems with minimal cost (High-level synthesis).

We designed a program to creation configuration library. It can be used as a static system configuration tuning for the most popular applications. We compared result of this program with standard dynamic system configuration tuning for system IBM SP/2 and we've got the result on 10-20% better.

We designed a program to synthesis of optimal structure reconfigurable scale distributed memory system based on ADSP-2106x chips for applications. An optimization criterion was minimal number of computational modules with the given parallel program execution time. Each module consists of fourth completely connected ADSP-2106x processors.

7. CONCLUSION

We presented to you our static list scheduling approach with the following main characteristics:

- approach is universal. It can be used for any topologies of computer systems;
- complex implementation of the task order formation and allocation procedure;
- communication cost model assumes the existence of an I/O processor;
- task can be assigned to the free or busy processor which provided minimal measure of initial execution time;
- the second pass allows a "compress" schedule of parallel program.

The future works are connected with solving the following tasks:

- comparison time complex of our and well known algorithms (Annex E and D of the Ada 95 Standard);
- problem of correct presentation parallel programs by the task graphs and its transformation;
- creation of hybrid scheduling approach (static+dynamic) with our heuristic;
- query optimization and optimal processor assignment in Parallel Relational Databases.

8. REFERENCES

[1] Coffman E., Graham R. Optimal scheduling for two-processor systems, Acta Informatica, Vol. 1, (1972).

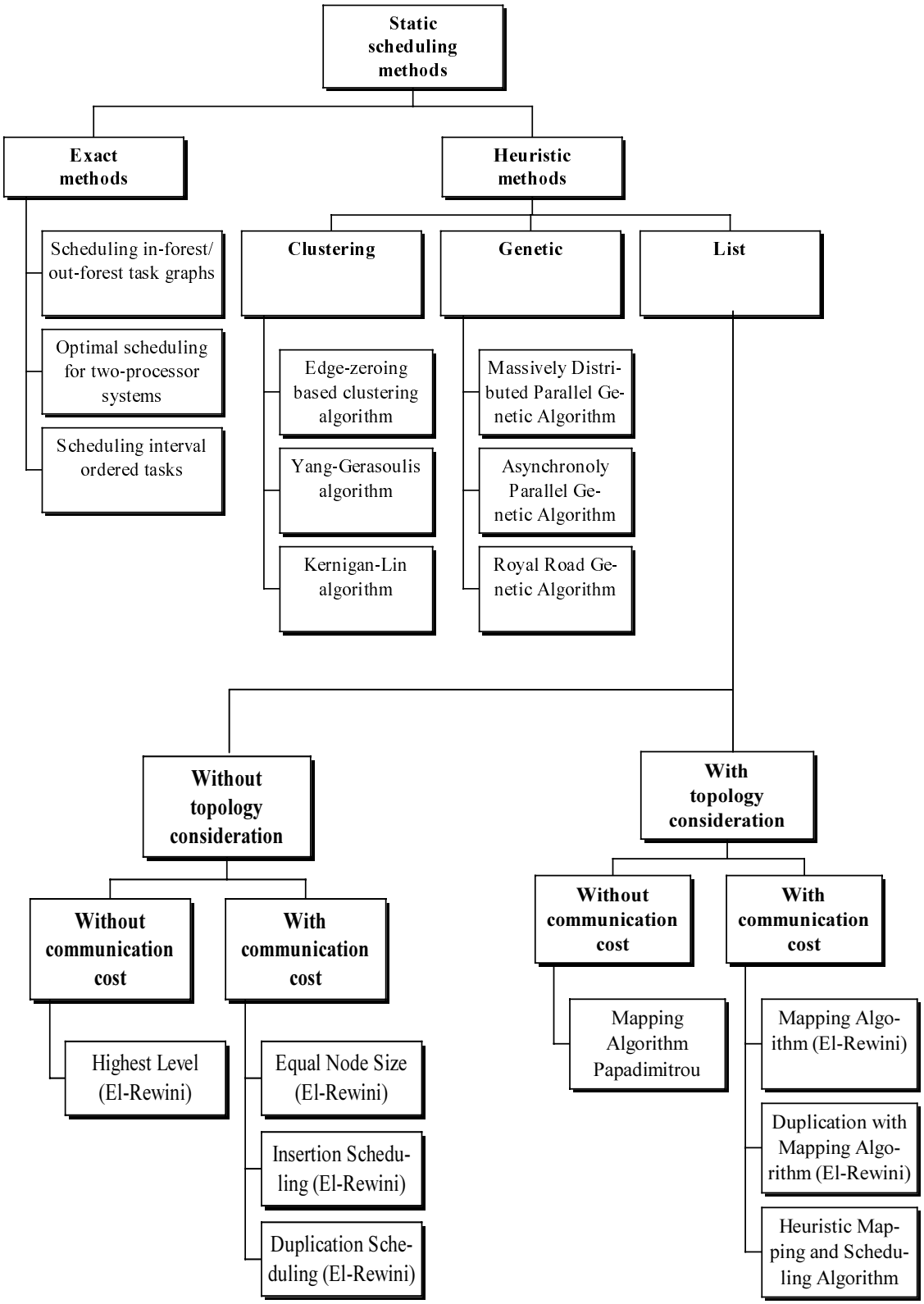
[2] Hesham El-Rvini, Levis T.G., Heshami A/ Task sheduling in distributed memory systems. Prenthall, 1994.

[3] Hesham El-Rebini, Lewis T.G. Parallel and distributed computing. Maining, 1998.

[4] Hu T., Parallel sequencing and assembly line problems, Operation Research, Vol. 9, (1981), 841-848.

[5] Papadimitriou C., Yannakakis M. Scheduling interval-order tasks, SIAM J.Comput., Vol. 8, N 3, (1979).

Appendix A


Static scheduling methods
Exact methods
Heuristic methods
Scheduling in-forest/ out-forest task graphs
Optimal scheduling for two-processor systems
Scheduling interval ordered tasks
Clustering
Genetic
List
Edge-zeroing based clustering algorithm
Yang-Gerasoulis algorithm
Kernigan-Lin algorithm
Massively Distributed Parallel Genetic Algorithm
Asynchronoly Parallel Genetic Algorithm
Royal Road Genetic Algorithm
Without topology consideration
With topology consideration
Without communication cost
With communication cost
Without communication cost
With communication cost
Highest Level (El-Rewini)
Equal Node Size (El-Rewini)
Insertion Scheduling (El-Rewini)
Duplication Scheduling (El-Rewini)
Mapping Algorithm Papadimitrou
Mapping Algorithm (El-Rewini)
Duplication with Mapping Algorithm (El-Rewini)
Heuristic Mapping and Scheduling Algorithm

Appendix B

Table 1. Performance measure of algorithm

A	1	2	3	4	5	6	7	8	9	10
50/50	1,17	1,25	1,33	1,54	1,78	1,85	1,92	2,23	2,51	2,54
60/40	1,14	1,18	1,29	1,45	1,60	1,71	1,83	2,11	2,07	2,10
70/30	1,09	1,13	1,21	1,29	1,34	1,17	1,31	1,51	1,62	1,64
80/20	1,05	1,14	1,15	1,19	1,21	1,29	1,30	1,32	1,35	1,34
90/10	1,01	1,10	1,12	1,15	1,15	1,17	1,19	1,19	1,20	1,19

Table 2. Completely connected topology

N Algorithm	8	16	32	64
MA	1.15	1.14	1.13	1.11
DMA	1.23	1.22	1.20	1.21
HMSA	1.08	1.09	1.07	1.05

Table 3. Ring topology

N Algorithm	8	16	32	64
MA	1.43	1.39	1.42	1.38
DMA	1.47	1.42	1.44	1.41
HMSA	1.36	1.30	1.35	1.33

Table 4. Star topology

N Algorithm	8	16	32	64
MA	1.44	1.43	1.41	1.41
MDA	1.56	1.53	1.52	1.57
HMSA	1.43	1.44	1.40	1.48

Table 5. Mesh topology

N Algorithm	8	16	32	64
MA	1.15	1.14	1.13	1.11
MDA	1.30	1.32	1.29	1.30
HMSA	1.25	1.25	1.25	1.24

Table 6. Tree topology

N Algorithm	7	15	31	63
MA	1.35	1.36	1.34	1.35
MDA	1.37	1.38	1.38	1.37
HMSA	1.29	1.28	1.26	1.27

Table 7. Hypercube topology

N Algorithm	8	16	32	64
MA	1.28	1.26	1.23	1.21
MDA	1.45	1.43	1.45	1.44
HMSA	1.28	1.25	1.23	1.22