Triplet : a Clustering Scheduling Algorithm for Heterogeneous Systems

Bertrand Cirou
LaBRI, Université Bordeaux I
351, cours de la Libération
33405 Talence Cedex, France
cirou@labri.fr

Emmanuel Jeannot
LORIA, Université Nancy I
615, rue du Jardin Botanique
54602 Villers les Nancy, France
ejeannot@loria.fr

Abstract

The goal of the OURAGAN project is to provide access of meta-computing resources to Scilab users. We present here an approach that consists, given a Scilab script, in scheduling and executing this script on an heterogeneous cluster of machines. One of the most effective scheduling technique is called clustering which consists in grouping tasks on virtual processors (clusters) and then mapping clusters onto real processors. In this paper, we study and apply the clustering technique for heterogeneous systems. We present a clustering algorithm called triplet, study its performance and compare it to the HEFT algorithm. We show that triplet has good characteristics and outperforms HEFT in most of the cases.

1 Introduction

Scilab is an heavily used tool in the mathematical community [7]. As Matlab, Scilab allows to execute scripts for engineering and scientific computation. However, Scilab has some limitation since it is not parallelized. The goals of Scilab$_{//}$[1], developed in the OURAGAN project[1]is to permit an efficient and transparent execution of Scilab on a meta-computing environment. Various approaches have been taken in order to achieve these objectives. The approach we propose is the following. Given a Scilab script, first, we analyze and compute its dependencies. Second, we build a task graph that model the inner parallelism of the script. This script is then scheduled on an heterogeneous cluster of workstations. In the last step we execute this script on the cluster. In this paper we focus on the scheduling and executing steps. In the literature a lot of work has been done for scheduling task graphs to an homogeneous set of processors [2, 4]. Algorithms for homogeneous processors are inefficient for most of network of workstations (NOWs). Indeed, most of the time, NOWs are made of heterogeneous computers. Several algorithms have been proposed to tackle the problem of scheduling tasks on an heterogeneous architecture [9, 10, 13]. All these algorithms implement the list-based scheduling technique. Two-step scheduling techniques have been shown to be very efficient for homogeneous systems [6, 11, 14, 15]. A two-step scheduling algorithm works as follows. The first step is the clustering phase : tasks are grouped into clusters. The main idea of this phase is to group tasks on virtual processors in order to suppress unnecessary communications. The second phase is called the mapping phase: each cluster is assigned a processor. This technique has been very successful because the clustering phase is global. This is opposed to list scheduling algorithms where only local optimizations are performed.

The research topic concerning clustering of static task graphs in the case of an heterogeneous platform is relatively unexplored. M. Eshaghian and C. Wu have proposed an algorithm called cluster-M in [5]. However, in our opinion, cluster-M has the following deficiencies. As the progression of the clustering is done by following the topological order of the task graph, bad clusters are then built. Moreover, this clustering always embeds the most communicant task onto its father, which is not always the best way to generate parallelism. Thus, the clustering computed by the cluster-M algorithm may not always be effective. In this paper, we propose a theoretical metric which describes the behavior of a good clustering algorithm. Our main contribution is that we propose a multi-step scheduling algorithm for heterogeneous NOWs. In order to apply the clustering technique to heterogeneous systems we show that we need to cluster both tasks and machines. We show that our algorithm, called *triplet* behaves as requested by the metric. Finally, we have compared our algorithm to the HEFT algorithm [9]. It appears that, in general, for heterogeneous network of workstations, triplet outperforms HEFT.

This paper is organized as follows. In Section 2, we describe the model of task graphs and heterogeneous systems we target. In Section 3, we present the new metric. Our

[1] `http://www.ens-lyon.fr/~desprez/OURAGAN`

multi-step algorithm is presented in Section 4. The metric conformity is shown in Section 5. The complexity of our algorithm is computed in Section 6. Experimental results are described in Section 7. In Section 8 we give concluding remarks.

2 Definitions and Models

Task Graphs. We use the task graph model to model our programs. A task graph is an annotated directed acyclic graph defined by the tuple $G = (V, E, T, C)$. V is the node set, representing a task. E is the edge set. There is an edge between task i and task j if there is a dependence between task i and task j. T is the number of instructions task set. C is the communication volume set. Transforming a Scilab script into a task graph is out of the scope of this paper. For more details the reader should refer to automatic parallelization techniques [3] or to the MATCH project [8].

Heterogeneous System Model. Heterogeneous systems we target are networks of workstations as one can find in a laboratory. Each workstation can communicate to any other workstation but communication links may have different speed. Each workstation may be different and can executes tasks at different speed. Hence we model an heterogeneous system by the following tuple : $H = (S, L)$ where S is the set of machine speed. The number of machines is $|S|$. L is the link bandwidth set. There is a communication link between every machine.

Execution Model. We assume that tasks are atomic: a machine executes a single task at a time. The total amount of CPU time required to execute a task is calculated by dividing the number of instructions of the task by the power of the CPU in MIPS. For instance, executing task i on machine m requires total amount of time of $T_{\text{comp}} = t_i/s_m$ where $t_i \in T$ and $s_m \in S$. A task can start its computations only when it has received all its data and can send data only when its computations are finished. The time taken to transfer data from task i to task j between machine m and n is $T_{\text{com}} = c_{i,j}/l_{m,n}$ where $c_{i,j} \in C$ and $l_{m,n} \in L$.

3 Metric

A metric allows to class algorithms depending on solutions they produce. A well known metric on homogeneous platform is the speedup, but it looses some sense when applied to the heterogeneous case. Hence we need to find new ones for evaluating algorithms in the general case. Yarmolenko et al. proposed in [16], new criteria called efficiency and utilization. This metric only apply to independent tasks and without communication. Here are the definitions for evaluating processors: let TET be the total execution time of all tasks for a given schedule and T_{seq} the total execution time of all tasks on the best processor. Efficiency and utilization for processors are defined as follow:

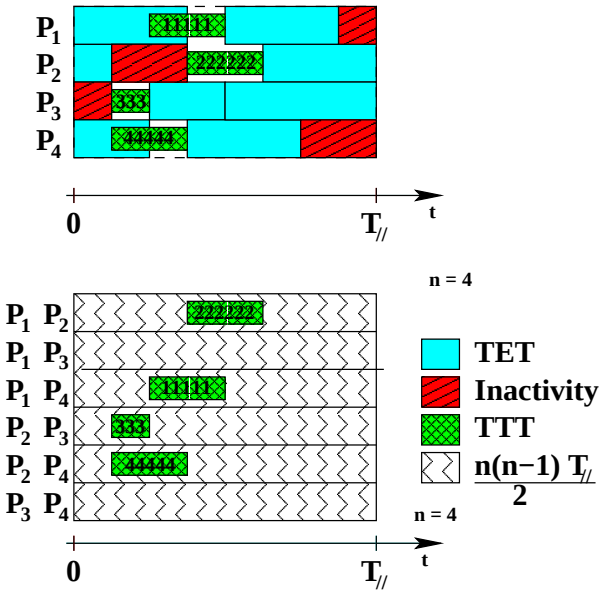

Figure 1. Gantt Chart for Processors and Network Links

$$E = \frac{T_{\text{seq}}}{TET}\ , U = \frac{TET}{nT_{//}}$$

Thus, $T_{//}$ can be extracted from these two equations:

$$T_{//} = \frac{T_{\text{seq}}}{nEU}$$

Since this metric does not take the network into account, we make an extension with two new formulas for evaluating communications.

In Figure 1 we present two Gantt charts, the upper one is for the tasks scheduling and the second below corresponds to the scheduling of the communications.

Let n the number of workstations, TTT be the total transfer time, TST the total of the n smallest communications time and $\frac{n(n-1)}{2}$ the number of links in the fully connected graph of processors.

TST is a constant for a given DAG and a given platform, this value is calculated by dividing the $n-1$ smallest communication by the bandwidth of the fastest link. TST is the minimal communication time spent when all the $n-1$ processors are used (the first task starts its computation on one processor and launches computations on the $n-1$ other processors with these $n-1$ communications). We set the

network efficiency and utilization for n workstations:

$$E_{\text{net}} = \frac{TST}{TTT}\ , U_{\text{net}} = \frac{2TTT}{n(n-1)T_{//}}$$

The network efficiency indicate whether only necessary communications are performed. The network utilization give an estimation of the average load of the network. If communications are present all along the execution of the program then we reach the case where the network utilization is maximal.

We can express $T_{//}$ as a function of the efficiency and the utilization. We get a new network dependent formula.

$$T_{//} = \frac{2TST}{E_{\text{net}}U_{\text{net}}n(n-1)}$$

Minimizing $T_{//}$ can be done by maximizing the product of the efficiency by the utilization. An important fact is that utilization and efficiency are divergent, the more utilization grows, the smaller efficiency is.

4 The Triplet Algorithm

Homogeneous versus Heterogeneous. Clustering algorithms have been very successful for homogeneous platforms [12, 15]. These algorithms are fast : for the best clustering algorithms the complexity is bounded by sorting edges of the task graph. Moreover, list-scheduling algorithms traverse the graph using a topological sort and therefore does only local optimizations. On the other hand, clustering algorithms consider global criterions to map tasks to clusters and then are able to perform global optimizations. Hence, it appears that in most of the cases, clustering algorithms are faster and give better results than list-scheduling algorithms. This remark motivates us for trying to build a clustering algorithm for heterogeneous platforms.

However, adapting clustering algorithms to the heterogeneous case is a difficult task. In homogeneous systems the duration of a communication depends only on the number of data exchanged and the duration of a task depends only on the number of operations to perform. In an heterogeneous system this is no longer true. Indeed, the duration of a communication depends also on the speed of the network link taken and the duration of a task depends on the processor that will execute this task. Therefore, techniques used for clustering tasks on homogeneous systems such as comparing the size of data exchanged between tasks or the number of operations of a task give little informations for an homogeneous system (large data can be exchanged rapidly on a fast link and small data can be exchanged slowly on a slow link). Hence, clustering algorithms for homogeneous systems cannot give good results on heterogeneous systems.

Since one cannot know if a given task or a given communication is going to be longer that an other task or communication prior to mapping clusters to processors, we propose to group machines that share the same characteristics (network, processor speed,etc...) and then to map clusters of task to clusters of machines. The main advantage of this approach is that these clusters of machines are sets of somehow homogeneous hardware. Hence, during the clustering phase, we can take decisions knowing that clusters of tasks are going to be mapped on relatively homogeneous clusters of machines.

Our multi-step scheduling algorithm for heterogeneous platforms is a bit different than multi-step scheduling algorithm for homogeneous platforms since it is performed in three steps.The first step is the clustering of tasks. Tasks are grouped into clusters in order to suppress unnecessary communications while preserving parallelism. The second step is the workstation clustering. As mentioned above, in order to efficiently map clusters to machines, machines which are somehow equivalent need to be grouped together. In the last step, task clusters are mapped to workstation clusters.

Algorithm 1 Tasks clustering

Require: A task graph and a system topology graph
Ensure: The clustering of the task graph
1: Put each task in a cluster.
2: Generate the list of all the triplets.
3: Sort the triplets by decreasing degree and by decreasing amount of communication.
4: **for** each triplet **do**
5: **if** geometric or temporal criterion is fulfilled **then**
6: merge the two clusters
7: **end if**
8: **end for**

Clustering Tasks. Algorithm 1 is our task clustering algorithm. Initially, tasks are put in different clusters. In order to suppress unnecessary communications we need to merge some clusters. For merging clusters our algorithm considers tasks which belong to a path of length 2 in the task graph. Every path of length 2 is composed of three tasks and is called a *triplet*. We consider triplets of tasks instead of pairs of tasks because there are many more triplets than pairs. Thus, our algorithm test merging possibilities more often along the growth of clusters.

Before starting the clustering phase, triplets are all generated. Then, they are all considered one at a time. Therefore, we need to sort triplets in order to consider large communicating edges first. Triplets are sorted first by their degree and second by their decreasing amount of communication produced by its three tasks. Hence, our algorithm will first

clusterize parts of a task graph that presents few parallelism and high communications costs.

Our algorithm considers each triplet. Let t_2 and t_4 be two tasks of a triplet with t_2 a predecessor of t_4 and respectively belonging to cluster C_1 and C_2 (see Figure 2). Cluster C_1 and C_2 are merged if one of the two following criteria is true.

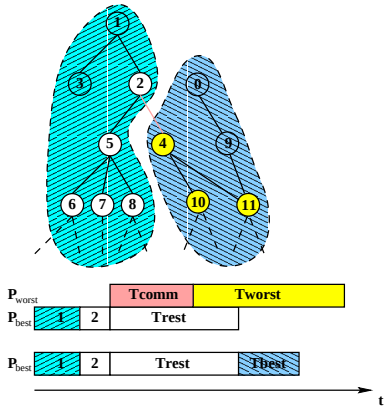

Figure 2. Temporal criterion

Figure 2 shows the first criterion that is based on temporal parameters. Let T_{rest} be the time needed to compute all successors of t_2 that are in C_1 on the best processor. Let T_{worst} be the time required to execute on the worst processor all successors of t_4 that are in C_2. Let T_{best} be the time to execute all tasks of C_2 on the best processor. Let T_{comm} be the duration of the communication between t_2 and t_4 evaluated on the worst network link. Our first criterion cause the merging when $T_{\text{comm}} + T_{\text{worst}} < T_{\text{rest}} + T_{\text{best}}$. This means C_1 and C_2 are not merged only if it worth not merging in the worst case. This criterion controls the width of cluster by suppressing parallelism each time there is a risk to be slower if the two clusters are on different processors. The second criterion use geometric properties of the clusters. Two clusters C_1 and C_2 are merged if they do not overlap more than 20% and the resulting cluster is at least 20% greater than initials clusters. This criterion is purely morphological, its main goal is to keep clusters elongated.

Clustering Workstations. After having done task clustering, we clusterize the workstation graph as well to get a global view of the problem. This clustering is needed for detecting machines that present the same characteristics. We suppose we deal with LAN type network, where each computer is able to do point to point communications. The clustering is done by sorting machines, then by going through the sorted list and creating a new cluster each time the variation between two consecutive workstations is big enough.

Algorithm 2 Mapping task's clusters onto clusters of workstations

Require: A set of task's clusters and a set of workstation's clusters

Ensure: Assign a Workstation to each task

1: Sort workstation's clusters by decreasing network capabilities and by CPU power.
2: Determine a maximum load for each cluster of workstation.
3: Sort task's clusters by amount of external communications and by number of instructions.
4: **for** each task's cluster in the order **do**
5: **if** if the number of operations assigned to current cluster of workstations exceed its load. **then**
6: switch to next workstation cluster.
7: **end if**
8: Assign current task's cluster to the Workstation having the best completion time.
9: **end for**

We sort workstations, first by decreasing network capabilities, then by decreasing CPU power. Clusters are defined in the following way: while two consecutive workstations have less than 10% of difference concerning their bandwidth and CPU power we add them on the same cluster. In the other case we create a new cluster with the last workstation considered. This clustering is fast and permits to put together computers that are somewhat equivalent.

Mapping Tasks Clusters onto Workstations Clusters. Our mapping algorithm is shown in Algorithm 2. Each task cluster has its own values for the amount of output communication and the total number of instructions required to achieve. Clusters of workstations are labeled with their overall network capabilities and total CPU power. Before doing the mapping, we need to sort clusters of each sort (first by the network parameter and second by the computation parameter). Moreover, the mapping must equitably load each cluster of workstations, hence we need the representation of each of these clusters compared with the others. Our mapping algorithm allocates task's cluster, in the order, to workstations having the best completion time as long as the load limit is not exceeded. This mapping insure that the largest communications are done on the best links and each cluster of processors has nearly the same time computation load.

5 Metric Conformity

The triplet algorithm contributes to minimize $T_{//}$. In the two formulas we got for $T_{//}$, we need to maximize the product of the efficiency by the utilization. E is improved at the

assignment time, because we choose the processor that minimize execution finish time of the cluster being mapped. U is maximized thanks to the load balancing done between clusters of workstations.

Concerning the network, the reduction of $T_{//}$ depends of the product $E_{net}U_{net}$. During the clustering, only communications that are profitable are kept. Thus, E_{net} is high. For U_{net}, the geometric criterion increase the number of tasks executable at time t. Hence, the probability of communications is high and then the value of U_{net} too.

6 Complexity

Let be Δ_i^+ and Δ_i^- the in and out degree of the task i and Δ_{max}^+, Δ_{max}^- the maximum in and out degree of the graph. Let be N_t the number of triplets in the DAG. Each task i contributes to $\Delta_i^+\Delta_i^-$ triplets : for one incoming edge there is one triplet per outgoing edge. Thus N_t is the sum of all these triplets for each task in the DAG.

$$N_t = \sum_{i=1}^{|V|} \Delta_i^+\Delta_i^- < \sum_{i=1}^{|V|} \Delta_{max}^+\Delta_{max}^- = |V|\Delta_{max}^+\Delta_{max}^-$$

In the worst case, $\Delta_{max}^+ = \Delta_{max}^- = O(|V|)$ and the number of triplets is $O(|V|^3)$. However for most of the graphs Δ_{max}^+ and Δ_{max}^- are constant and small. For some graphs only the maximum out degree *or* the maximum in degree is related to $|V|$ (In the Gaussian Elimination task graph, for instance $\Delta_{max}^+ = 2$ and $\Delta_{max}^- = 0(\sqrt{|V|})$. Let $\Delta_{max} = \Delta_{max}^+\Delta_{max}^-$. The complexity of the clustering phase is bounded by a sort of all the triplets, hence its complexity is $O(|V|\Delta_{max}\log(|V|\Delta_{max}))$.

Let p be the number of processors. For the mapping step, the worst case is reached when we have only one cluster of tasks and one of workstations. For each task, the best processor is taken among the p ones available. The mapping takes $O(p|V|)$. The triplet algorithm takes $O(|V|\Delta_{max}\log(|V|\Delta_{max}) + p|V|)$.

Since for most of the task graphs Δ_{max} is a constant, we claim that our algorithm has a very competitive complexity.

7 Results

We have implemented a task graph execution simulator for testing various heterogeneous topology. We use another task scheduling algorithm: HEFT [9] (Heterogeneous Earliest Finish Time) to make a performance comparison with our triplet algorithm. The HEFT algorithm is based on evaluating the shortest path of execution to a terminal task.

We define the processor heterogeneity and network heterogeneity as follows:

$$H_{\text{proc}} = \frac{\text{standard deviation of CPU power}}{\text{average of CPU power}}$$

$$H_{\text{net}} = \frac{\text{standard deviation of network bandwidth}}{\text{average of network bandwidth}}$$

If we take 10 PCs at 333Mhz, 5 PCs at 800Mhz and 5 PCs at 1GHz, then $H_{\text{proc}} \sim \frac{292.18}{616.5} \sim 43.4\%$ If we replace the 10 PCs at 333Mhz by others at 166MHz, the heterogeneity reaches 70%

These definitions allow to compare the heterogeneity of recent workstation network with older ones, because we have the average value in the formula that has a normalization effect. For our benchmarks we have generated $40\ 000$ task graphs, and $40\ 000$ different topologies with varying heterogeneity. We have fixed the heterogeneity of the bandwidth at 50%, on the other hand the processor heterogeneity takes the following values: 31%, 36%, 42%, 48%. Values indicated in Figure 3 correspond to the middle of the interval inside which each parameter is randomly chosen. We expose here only a part of our results, we have in fact 16 histograms with various task graph heterogeneity. As they are quite similar we chose not to present them. The y-axis of the results shown Figure 3 is the time our solution is faster than HEFT solution : we outperform HEFT each time the bargraph is greater than 1. At the first look we see that the performance of our triplet algorithm increases with the heterogeneity. Our algorithm manages the heterogeneity better than HEFT does, thanks to our multi-step approach: clustering then mapping. We have evaluated our metric by checking wether or not there could be schedules that have poor efficiency and utilization but produce good makespan. We conducted 3000 tests for HEFT and triplet on various DAGs of different heterogeneity and got the result that the algorithm that have the best makespan always have the best product of efficiency by utilization.

8 Conclusion and Future Work

We have presented a new algorithm for scheduling task graph on a heterogeneous system of workstations. Our contribution is three-fold. First, The algorithm is based on a multi-step approach that allows global optimizations. Second, we have presented a new metric that express requirements a good scheduling algorithm must have and we have shown that our algorithm fulfills these requirements. Lastly, we have compared our algorithm to the well-known and efficient HEFT algorithm. In most of the cases, our algorithm outperforms HEFT.

We want our execution scheme to adapt to load unbalance that appears when other users run applications on the cluster. In order to do that we plan to migrate tasks within its machine clusters – a machine cluster is composed of similar machines – during execution.

Finally, we plan to incorporate this algorithm to Scilab$_{//}$ in order to execute Scilab script on clusters of workstations.

References

[1] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubi, S. Steer, F. Suter, and G. Utard. Scilab to Scilab$_{//}$, the OURAGAN Project. *To appear in Parallel Computing*, 2001.

[2] P. Chretienne and C.Picouleau. *Scheduling Theory and its Applications*, chapter 4, Scheduling with Communication Delays: A Survey, pages 65–89. John Wiley and Sons Ltd, 1995.

[3] M. Cosnard and M. Loi. Automatic Task Graph Generation Techniques. *Parallel Processing Letters*, 5(4):527–538, 1995.

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

[5] M. M. Eshagian and Y. C. Wu. Mapping heterogeneous task graphs onto heterogeneous system graphs. *Heterogeneous Computing Workshop (HCW'97)*, pages 147–160, April 1997.

[6] A. Gerasoulis and T. Yang. On the Granularity and Clustering of Direct Acyclic Task Graphs. *IEEE Transactions on Parallel and Distributed Systems*, 4(6):686–701, June 1993.

[7] C. Gomez, editor. *"Engineering and Scientific Computing with Scilab"*. Birkhäuser, 1999.

[8] M. Haldar, A. Nayak, A. Kanhere, P. Joisha, N. Shenoy, A. Choudhary, and P. Banerje. Match Virtual Machine: An Adaptive Runtime System to Execute MATLAB in Parallel. In *International Conference on Parallel Processing (ICPP-2000)*, Toronto, Canada, Aug. 2000.

[9] S. H. Haluk Topcuoglu and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. *8th IEEE Heterogeneous Computing Workshop (HCW'99)*, pages 3–14, April 1999.

[10] M. Kafil and I. Ahmad. Optimal task assignment in heterogeneous computing systems. *Heterogeneous Computing Workshop*, pages 135–146, April 1997.

[11] Y.-K. Kwok and I. Ahmad. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521, May 1996.

[12] J.-C. Liou and M. A. Palis. A New Heuristic for Scheduling Parallel Programs on Multiprocessor. In *IEEE Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT'98)*, pages 358–365, Paris, Oct. 1998.

[13] A. Radulescu and A. van Gemund. Fast and effective task scheduling in heterogeneous systems. In *Proceeding of Heterogeneous Computing Workshop*, 2000.

[14] V. Sarkar. *Partitionning and Scheduling Parallel Program for Execution on Multiprocessors*. MIT Press, Cambridge MA, 1989.

[15] T. Yang and A. Gerasoulis. DSC Scheduling Parallel Tasks on an Unbounded Number of Processors. *IEEETPDS*, 5(9):951–967, Sept. 1994.

[16] V. Yarmolenko, J. Duato, D. K. Panda, and P. Sadayappan. Characterization and Enhancement of Static Mapping Heuristics for Heterogeneous Systems. Technical Report OSU-CISRC-02/00-TR07, Dept. of computer science, Ohio State University, Feb. 2000. To be presented in HiPC'2000.

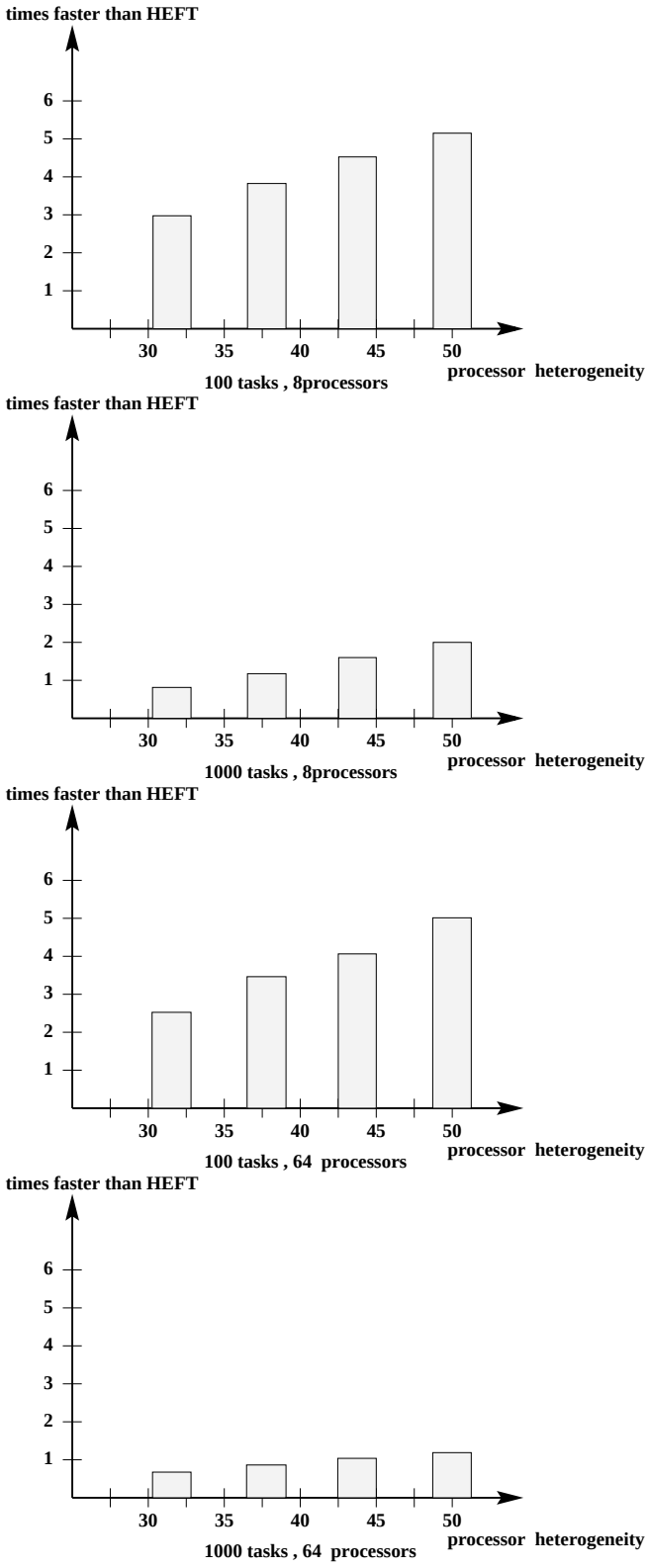

Figure 3. Average values: Communication 50Ko, Task $500\ 000$ instructions, Bandwidth 10 Mbits/s, CPU 300MHz