INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Triplet : a Clustering Scheduling Algorithm for Heterogeneous Platforms

Bertrand Cirou — Emmanuel Jeannot

N° 0248

Février 2001

THÈME 1

Rapport technique

ISSN 0249-0803

Triplet : a Clustering Scheduling Algorithm for Heterogeneous Platforms

Bertrand Cirou*, Emmanuel Jeannot †

Thème 1 — Réseaux et systèmes
Projet Résédas

Rapport technique n° 0248 — Février 2001 — 11 pages

Abstract: In this report we present a new scheduling algorithm for heterogeneous plateforms. It uses the clustering techniques and has a better behavior than the HEFT algorithm in most of the cases

Key-words: Scheduling, heterogeneous, clustering, HEFT

* LaBRI, Université de Bordeaux I
† LORIA, Université H. Poincaré

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Triplet : un algorithme de regroupement pour plates-formes hétérogènes

Résumé : Dans ce rapport nous présentons un nouvel algorithme d'ordonnancement pour plates-formes hétérogènes. Il utilise la technique du regroupement et se comporte mieux que HEFT dans la plupart des cas.

Mots-clés : Ordonnancenent,hétérogène, regroupement, HEFT

1 Introduction

Using network of workstations (NOWs) for scientific computation is not a new idea [4]. It is a cheap way for obtaining a parallel platform. In the literature a lot of work has been done for scheduling task graphs to homogeneous set of processors [1, 2]. This approach is not realistic for most of 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 [6, 7, 9]. All these algorithms implement the list-based scheduling technique. Two-step scheduling techniques have been shown very efficient for homogeneous systems [5, 8, 10, 11]. 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. On the contrary list scheduling algorithms which perform local optimizations.

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 such an algorithm called cluster-M in [3]. 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 [6]. It appears that, in general, for heterogeneous network of workstations, triplet outperforms HEFT.

This paper is organized as follows. Section 2 we describe the model of task graphs and heterogeneous systems we target. Section 3, we present the new metric. Our multi-step algorithm is presented section 4. The metric conformity is shown section 5. Experimental results are described section 6. In section 7 we give concluding remarks.

2 Definitions and Models

The task graph. 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.

The 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.

The 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

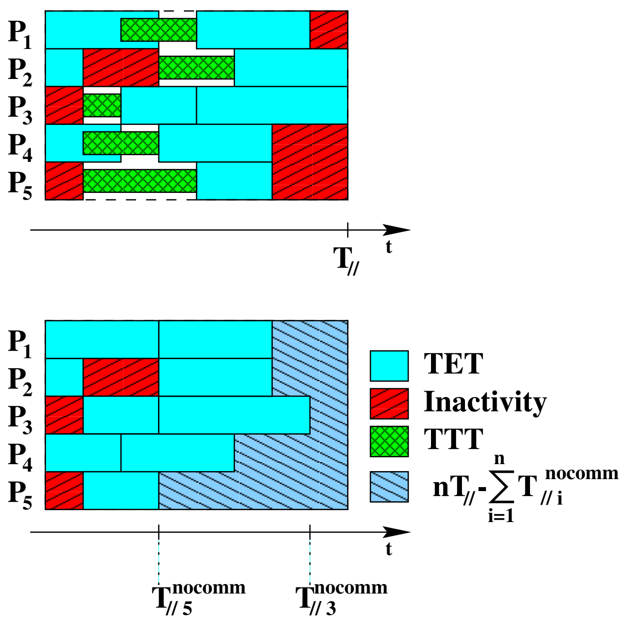

Figure 1: Gantt chart with and without communication

A metric allows to class algorithms depending on solutions they produce. A well known metric on homogeneous platform is the speedup, but it loose 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 [12], new criterions 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:

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

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

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

As this metric doesn't take account of the network, 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 real scheduling whereas the second below corresponds to a scheduling where the communication are done instantaneously. We introduce $T_{//i}^{\text{nocomm}}$ for each processor to measure the gap between these two cases.

Let TTT be the total transfer time, TST the total of the n smallest communications time and $T_{//i}^{\text{nocomm}}$ the completion time without considering communication on the i^{th} processor. 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 processors are used (the first task starts its computation on one processor and lauches computations on the $n-1$ other processors with these $n-1$ communications). We set the network efficiency and utilization for n workstations:

$$E_{\text{net}} = \frac{TST}{TTT} \text{ , } U_{\text{net}} = \frac{TTT}{nT_{//} - \sum_{i=1}^{n} T_{//i}^{\text{nocomm}}}$$

The network efficiency indicate whether only necessary communications are performed. The network utilization give an estimation of the overlapping of communications by computations. If they are all overlapped then we reach the case where no communication apparently occurs and the network utilization is maximal ($U_{net} = 1$).

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

$$T_{//} = \frac{TST}{nE_{\text{net}}U_{\text{net}}} + \frac{\sum_{i=1}^{n} T_{//i}^{\text{nocomm}}}{n}$$

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

We present here a multi-step algorithm for scheduling tasks on an heterogeneous system. 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. In order to map efficiently clusters to machines, machines which are somehow equivalent need to be grouped together. The last step is the mapping itself.

Algorithm 1 Tasks clustering

Require: A task graph and a system topology graph
Ensure: The clustering of the task graph

```
Put each task in a cluster.
Generate the list of all the triplets.
Sort the triplets by decreasing degree and by decreasing amount of communication.
for each triplet do
   if geometric or temporal criterion is fulfilled then
      merge the two clusters
   end if
end for
```

Clustering Tasks. Our clustering algorithm is shown Figure 1. 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 3 in the task graph. Every path of length 3 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. These triplets are defined as suited: a triplet is a path of length 3 in the task graph. Their number is bounded by $|E|^2$ (proof: consider the worst case of a maximum connected task graph). The complexity of the triplet algorithm is mainly conditioned by their number. An approximate value of this number can be get with the average in-degree (Δ^{in}) and out-degree (Δ^{out}) of the task's task graph. In fact the higher the average degree is, the more the number of triplets is significant because each task of the task graph is present in $\Delta^{in}\Delta^{out}$ triplets. As we have to sort them, the complexity of our algorithm can be expressed as $\Delta^{in}\Delta^{out}|V|\log(\Delta^{in}\Delta^{out}|V|)$. In task graphs corresponding to real parallel program the average degree is very often less than 10 so we can reformulate the complexity as $C|V|log(|V|)$.

We 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 criterion is true.

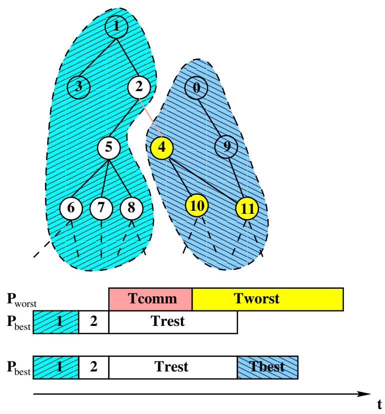

Figure 2: Temporal criterion

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 the sum of all tasks computations of C_2. 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's 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. 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

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

```
Sort workstation's clusters by decreasing network capabilities and by CPU power.
Determinate a maximum load for each cluster of workstation.
Sort task's clusters by amount of extern communications and by number of instructions.
for each task's cluster in the order do
  if if the number of operations assigned to current cluster of workstations exceed its
  load. then
    switch to next workstation cluster.
  end if
  Assign current task's cluster to the Workstation having the best completion time.
end for
```

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. As well 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 the workstation 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 overlapping is high and the value of U_{net} get closer to 100%.

6 Results

We have implemented a task graph execution simulator for testing various heterogeneous topology. We use an other task scheduling algorithm: HEFT [6] (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:

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

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

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. At the first look we see that the performance of our triplet algorithm increases with the heterogeneity. Our algorithm manage the heterogeneity better than HEFT do, thanks to our multi-step approach: clustering then mapping.

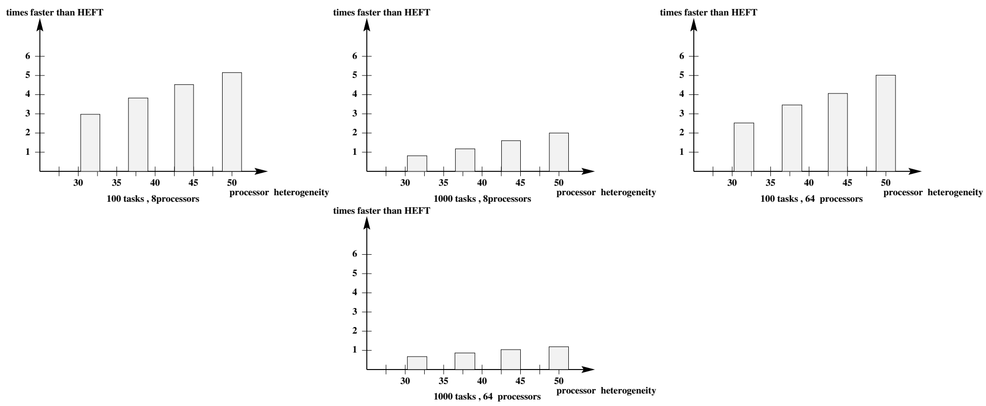

Figure 3: Average values: Communication 50Ko, Task 500 000 instructions, Bandwidth 10 Mbits/s, CPU 300MHz

7 Conclusion

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 fulfill these requirements. Lastly, We have compared our algorithm to the well-known and efficient HEFT algorithm. In most of the cases our algorithm outperform HEFT.

References

[1] 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.

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

[3] 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.

[4] A. Geist, A. Berguelin, W. Jiang J. Dongarra, R. Manchek, and V. Sunderam. *PVM3 User's Guide and Reference Manual.* ORNL, TM-12187 edition, September 1994.

[5] 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.

[6] Salim Hariri Haluk Topcuoglu and Min-You Wu. Task scheduling algorithms for heterogeneous processors. *8th IEEE Heterogeneous Computing Workshop (HCW'99)*, pages 3–14, April 1999.

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

[8] 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.

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

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

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

[12] 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, February 2000. To be presented in HiPC'2000.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803