Computación y Sistemas Vol. 4 No. 2 pp.158 -165pp
© 2000, CIC - IPN. *ISSN 1405-5546* Impreso en México

Scheduling Malleable Tasks with Convex Processing Speed Functions

Jacek Blazewicz[1], Maciej Machowiak[1], Jan Weglarz[1], Gragory Mounie[2] and Denis Tristram[2]
[1] Instytut Informatyki Politechniki Poznanskiej, Poland
[2]Laboratory Informatique et Distribution,
IMAG, Grenoble, France
E-mail: maciej.machowiak@cs.put.poznan.pl

Article received on February 15, 2000; accepted on August 23, 2000

Abstract

In the paper, the problem is considered of scheduling very large applications on parallel computer systems. These applications can be modeled as malleable tasks, i.e. tasks which processing speed depends on a number of processors granted. Using this concept it is proved analytically that for convex curves relating processing speed to a number of processors, gang scheduling strategy is optimal from the viewpoint of schedule duration.

Keywords: malleable task, convex processing speed function, gang scheduling.

1 Introduction

We consider a set of n independent tasks, each of them requiring for its execution one or more processors among a set of m identical parallel processors. The processors are fully connected by an interconnection network. The allotment of the processors influences the execution speed of a task (program), i.e. the more processors the faster the processing of a program.

The dependence of task processing times on a number of processors alloted, leads to the so called *malleable tasks (MT)*. A malleable tasks needs one or more processors for its execution, but the number of them is unknown in advance, only a function describing this dependence is known. This property distinguishes malleable tasks from the multiprocessor tasks, considered by Błażewicz et al. [1986] and [Lloyd, 1982], where the number of processors alloted to each task is known. The latter model has received a considerable attention in the literature. The problem of scheduling independent MT without preemption (it means that each task is computed on a constant number of processors from its start to completion) was studied in [Turek et al., 1992, Ludwig, 1995]. The problem is NP-hard [Du and Leung, 1989], thus, an approximation algorithm with performance guarantee has been looked for. While the problem has an approximation scheme for any fixed value m, no practical polynomial approximation better than 2 is known [Ludwig, 1995]. The 2-approximation presented in [Ludwig, 1995] is based on a clever reduction of MT scheduling to the 2 dimensional bin-packing problem, using the earliest result of [Turek et al., 1992] that any λ-approximation for the bin-packing problem can be polynomially transformed into a λ-approximation for the MT scheduling. Based on the above results Rapine, Mounie and Trystram [1999] have found a 2-phases algorithm with performance guarantee $\sqrt{3}$. In [Rapine et al., 1998] on-line scheduling strategies for time/space sharing were considered.

In [Schwiegelshohn et al., 1998] it has been presented an approximation algorithm for a system of independent tasks to be scheduled without preemption on a parallel computer with a minimum weighted average response time criterion. Prasanna and Musicus [1991] have considered a set of parallel tasks and associated precedence constraints, a finite pool of processor resources, and specified processing speed functions for each task as a function of applied processing power. In the special case where the processing speed function of each task is p^α, where p is amount of processing power applied to the task, $0 < \alpha < 1$, a closed form solution for task graph formed from parallel and series connection was derived.

In this paper we propose a model of malleable tasks in the form of a continuous function: processing speed vs. number of processors. Using this model one can utilize known results from the continuous resource allocation theory for studying the properties of optimal schedules. It will be proved that in the case of convex functions, the optimal (i.e. minimum length) schedules are built by a sequential performance of all tasks, each of them using a full power of the computing system. This proves also optimality of the gang scheduling strategy for very large applications like Cholesky factorization and out-of-core computation. In a practical sense, by application we will mean a task which is composed of the interacting processes (modules).

An organization of the paper is as follow. In Section 2 the scheduling problem is formulated, and in Section 3 its validity is discussed. In Section 4 main features of optimal schedules are proved for convex function relating processing speed of a task to a number of processors alloted. Section 5 discusses optimality of gang scheduling in the above context, Section 6 presents an example of the application of the proposed approach, and Section 7 concludes the paper.

2 Problem formulation

We consider a set of m identical processors $\mathcal{P} = \{P_1, P_2, ..., P_m\}$ used for executing the set $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ of n independent, nonpreemptable malleable tasks (MT). Each MT needs for its execution at least 1 processor but less than m. The number of processors alloted to a task is unknown in advance. The *processing speed* of a task depends on the number of processors alloted to it: namely, function f_i relates processing speed of task T_i to a number of processors alloted. The criterion assumed is schedule length. Let us note that processing times of MT's are sometimes represented by some factor μ, which determines the loss of time during a task processing using more than one processor, caused by communication delays or by synchronization needs. This factor, called *inefficiency factor*, is a discrete function of a number of processors and a type of a task and its geometrical interpretation is given in Figure 1.

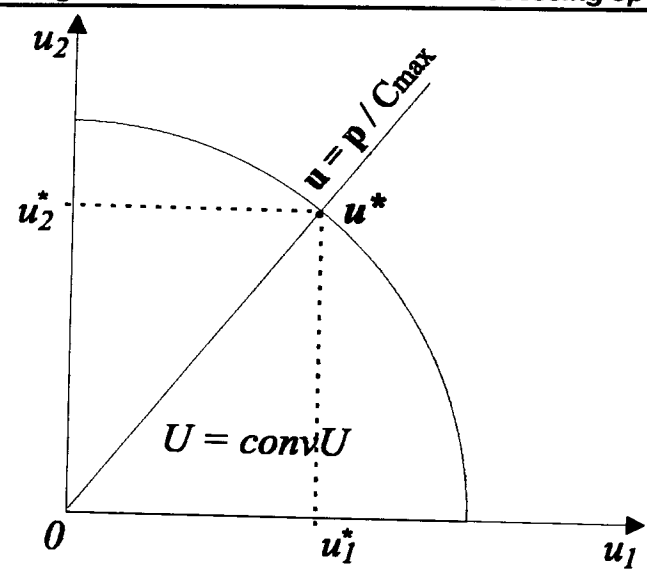


Figure 1: Inefficiency factor.

The relation between the speed function and the inefficiency factor is as follows:

$$\mu_i(r) = \frac{r t_i(r)}{t_i(1)} \rightarrow t_i(r) = \frac{t_i(1)}{r}\mu_i(r), \text{ or}$$

$$\frac{t_i(1)}{t_i(r)} = \frac{f_i(r)}{f_i(1)}. \text{ Thus, } f_i(r) = \frac{r f_i(1)}{\mu_i(r)}, \qquad (1)$$

where: r - a number of processors used, $r \in (0, m]$; $t_i(r)$ - processing time of task T_i on r processors; $t_i(1)$ - processing time of T_i on one processor; μ_i - inefficiency factor (a discrete function of r) for task T_i; f_i - processing speed function for T_i.

In this paper we consider a special case of scheduling independent MT, namely for convex processing speed funtions. This case corresponds to some practical applications where superlinear dependence of the processing speed on a number of processors granted (i.e. speed-up) can be achieved [Gustafson, 1988], like out-of-core computations or Cholesky factorization. For this case, using a continuous $f_i(r)$ for modelling MT's, we prove optimality of the gang scheduling strategy.

3 Practical interpretation of convex processing speed functions

The convexity of the speed function may appear unrealistic in the context of the parallel execution of computations, since, it means that applications allow superlinear speed-up. Such a behavior of an application is to be expected if the computations use other resources in addition to the processing power of the involved processors. The main resource used in addition to the processing power is the memory. In any modern computer, the memory is organized as a hierarchy of successive layers of physical memories where the size of the memory is inversely proportional to its speed: from the registers in the processors, through the one or two levels of cache memory, the physical RAM, to the swapped memory on disks.

We describe now two of such applications where superlinear speed-up can be achieved: namely Cholesky factorization and molecular dynamics.

3.1 The effect of cache misses in Cholesky factorization

Matrix computations are basic routines in most scientific applications, using fully the high performance pipelines of modern processor floating point arithmetics. The main problem is to feed the pipeline with the data.

We choose to illustrate the convex processing speed functions on one of the most popular numerical kernels, namely, the Cholesky factorization [Golub and Van Loan, 1996]. The speed of computations depends in this case on the fact whether or not a matrix fits into the cache memory [Dongarra et al., 1999]. Let us assume that the whole matrix does not fit into the cache and thus, it slows down the computation on one processor. When the number of processors increases, the size of the blocks (a line or a column) of the matrix used in the computation of a processor decreases. When the blocks fit into the cache, the speed of the computation becomes more important than the ratio of additional processor resources [Barrett et al., 1993]. This leads to superlinear dependence of the processing speed on a number of processor alloted (speed-up). The same effect occurs when the matrix does not fit into physical memory and thus some parts of the matrix are swapped on disks.

Figure 2 shows the time achieved in the execution of Cholesky routine of scalapack [Dongarra et al., 1999] on an IBM SP for a large matrix.

We see, that the computation does not complete on less than 4 nodes. The time reported in Figure 2 shows superlinear processing speed-up. This particular behavior is emphasized in Figure 3: the work is defined as the time needed to complete the task times the number of processors involved in the computations. In Figure 3, the work decreases when the number of involved processors increases. Thus, there is a superlinear dependence of the speed function on a number of processors alloted.

3.2 Molecular dynamics and out-of-core computations

The simulation of molecular dynamics is one of the most challenging problem in science. It is modeled as a N body problem. The computation of atom movements is irregular if interactions are spatially limited (cut-off). An efficient execution requires advanced technques allowing to overlap communication by computation like asynchronous buffered communications and multithreading. In the cases of protein behavior, computations may require to compute interactions between hundreds of thousands of atoms [Bernard et al., 1999]. Needless to say, such an execution needs a large memory.

On some of the top parallel computers, like Cray T3E, in order to simplify hardware and optimize communications, there is no virtual memory management, thus the available memory is strongly limited. Thus, when the instance of the problem does not fit into the memory of a processor, the execution cannot be performed directly. To complete the execution, the virtual memory management needs to be done "by hand" using out of core computations, that is loading and storing intermediate computations on a disk. Of course, this increases the time of an execution. Thus, when the number of processors is sufficient for storing the whole data in the memory of these processors, a superlinear speed-up will be observed.

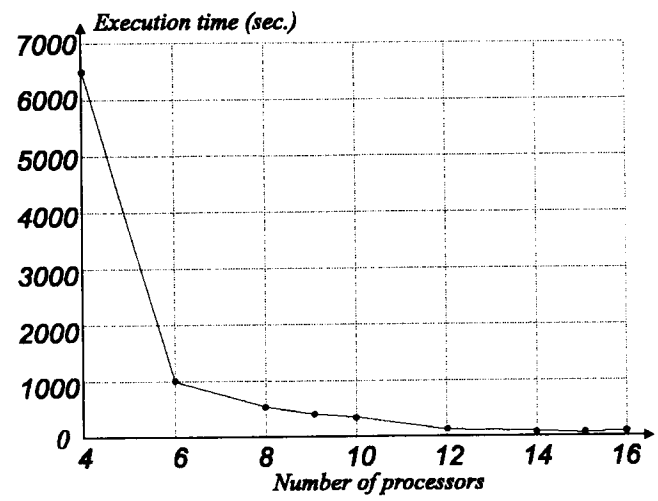


Figure 2: Execution time of scalapack cholesky, 7000x7000 matrix with bloc size 100x100, on IBM SP- 1.

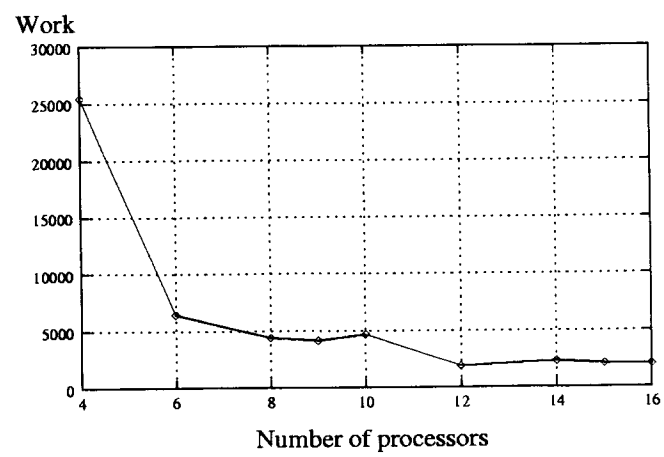


Figure 3: Work of scalapack cholesky, 7000x7000 matrix with bloc size 100x100, on IBM SP1.

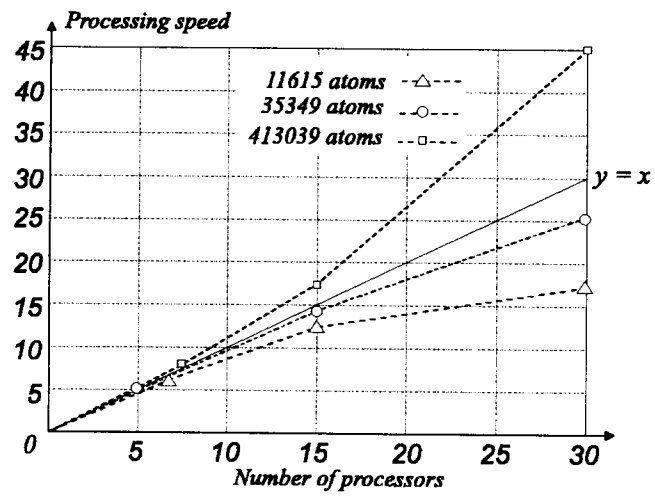


Figure 4: Function: a processing speed vs. a number of processors.

Figure 4 presents dependencies of a processing speed of large molecular dynamics simulation on a number of alloted processors for different problem sizes : from 11615 to 413039 atoms [Bernard, 1997]. The baseline $y = x$ is also drawn. The largest simulation does not execute on less than 8 processors. In order to accelerate computations, lists of interactions of atoms need to be kept. On 8 processors, these lists do not fit completely on the available memory. As the number of processors increases, more interactions may be stored, accelerating computations, and thus involving the convex behavior of the processing speed function. This convexity occurs until a threshold and then the processing speed becomes again concave due to communication and load-balancing overhead for more than 32 processors.

As we can see for some very large applications superlinear behavior of processing speed functions is achieved. It means that the shapes of functions (curves) are convex.

4 Basic results from optimal continuous resource allocation

In this Section, we will assume for the moment that functions relating a processing speed of a task (application) to a number of processors assigned to it are continuous. Then, using some results from the resource allocation theory [Węglarz, 1982], it will be proved that for convex functions a sequential processing of the tasks, each of which uses all processors available in the system, leads to a schedule of minimum length.

Assume that a processing of task T_i is described by the following equation:

$$\dot{x}_i(t) = dx_i/dt = f_i(r_i(t)),$$
$$x_i(0) = 0,\ x_i(C_i) = p_i \qquad (2)$$

where

- $x_i(t)$ - the state of T_i at moment t (i.e. the amount of processing done so far),
- $r_i(t)$ - a real number of resource units alloted to T_i at time t,
- f_i - a continuous, non-decreasing function, $f_i(0) = 0, f_i(r_i) > 0$,
- C_i - unknown in advance, finishing time of T_i,
- p_i - the final state or processing demand of T_i.

The total available resource amount is equal to m, i.e.

$$\sum_{i=1}^{n} r_i(t) \leq m \text{ for every } t$$

From (2) we have:

$$\int_0^{C_i} f(r_i)dt = p_i \qquad (3)$$

and thus $p_i = C_i f_i(r_i)$ or $C_i = p_i/f_i(r_i)$.

Denote by R the set of feasible resource allocations. Further, denote by U the set defined in the following way: $\mathbf{u} = (u_1, u_2, ..., u_n) \in U$ iff $\mathbf{r} \in R$, where $u_i = f_i(r_i)$. Elements of U will be called *transformed resource allocations.*

Theorem 4.1 [Węglarz, 1982]

The minimum schedule length for a set of n independent tasks described by (2) can always be expressed by the formula:

$$C^*_{max}(\mathbf{p}) = C^*_{max}(\mathbf{p}, U) =$$
$$= min\{C_{max} > 0 : \mathbf{p}/C_{max} \in convU\} \qquad (4)$$

where $\mathbf{p} = \{p_1, p_2, ..., p_n\}$ *is the vector of processing demands of tasks, and convU denotes the convex hull of U, i.e. the set of all convex combinations of elements of U.*

Proof Let us firstly prove the thesis for a step transformed resource allocation $\mathbf{u}$ equal to u_l, $l = 1, 2, ..., h$ in h disjoint time intervals included in $[0, C_{max}]$ with length equal to Δ_l , respectively,

$\sum_{l=1}^{h} \Delta_l = C_{max}$

Then (3) can be written in the form

$\sum_{l=1}^{h} u_{li}\Delta_l = p_i$

or, for the set T of tasks,

$\sum_{l=1}^{h} u_l\Delta_l = \mathbf{p}$, or $\sum_{l=1}^{h} u_l\Delta_l/C_{max} = \mathbf{p}/C_{max}$.

Putting $\lambda_l = \Delta_l/C_{max}$, we have

$\sum_{l=1}^{h} u_l\lambda_l = \mathbf{p}/C_{max}$, $\lambda_l \geq 0$, $l = 1, 2, ..., h$, $\sum_{l=1}^{h} \lambda_l = 1$

This means that the same schedule length C_{max} can be obtained using the constant transformed resource allocation $\mathbf{u}^{\cdot}$, which is the intersection point of straight line $\mathbf{u} = \mathbf{p}/C_{max}$ and the set of all convex combinations of u_l, $l = 1, 2, ..., h$.

Taking into account (2) and (4), as well as the fact f_i, are non-decreasing, we come to the conclusion that the minimum schedule length C^*_{max} can always be determined by the intersection point $\mathbf{u}^{\cdot *}$ (not necessarily being a transformed resource allocation!) of straight line $u_i = p_i/C_{max}$, $i = 1, 2, ..., n$ and the boundary of the set of all convex combination of set U, i.e. the boundary of set *convU*. That means that (4) holds. □

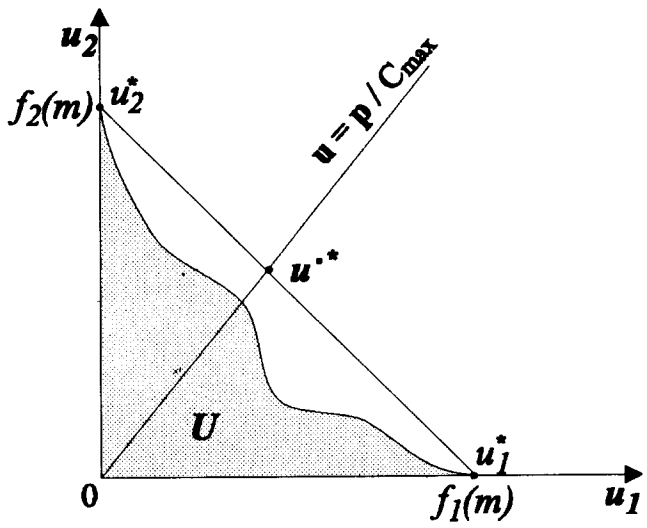


Figure 5: Set U and *convU*.

Now we describe geometrical interpretation of the problem (see Figure 5). From the above Theorem we know that the minimum schedule length is determined by the intersection point of straight line $u_i = p_i/C_{max}$, $i = 1, 2, ..., n$ and the boundary of set *convU*. This means that the shape of boundary of *convU* is of basic importance for the form of an optimal schedule. Because set U results from a transformation of set R using functions f_i, only these functions f_i decide about the shape of set U, and thus *convU*.

5 Gang scheduling and its optimality

Gang scheduling is a concept introduced by Ousterhout [1982]. He observed that a MIMD (Multiple Instruction Multiple Data) system performance degrades when a parallel application (task) does not have all its interacting processes scheduled at the same time. Gang scheduling (called also co-scheduling) consists in granting simultaneously (in the same time quantum) the processors to the same task. It has been demonstrated experimentally in [Feitelson and Rudolph, 1992] that gang scheduling performs well in a wide range of conditions and for various models of parallel tasks. Some practical examples describing a concept of gang scheduling can be found in [Drozdowski, 1996][Scherson et al., 1996].

Below we will show analytically that in case of malleable tasks and convex processing speed functions gang scheduling is an optimal strategy (i.e. leads to a minimum length schedules).

Corollary 5.1

For convex f_i, $i = 1, 2, ..., n$, the processing of individual tasks consecutively on all available processors (i.e. so called *gang scheduling*) is optimal.

Proof For convex f_i ($n = 2$) set U has the shape like the one presented in Figure 6. The intersection point $\mathbf{u}^{\cdot *}$ of straight line $u_i = p_i/C_{max}$ and the boundary of the set *convU* is never a transformed resource allocation (except where f are linear), but the same (minimum) schedule duration is also obtained using transformed resource (processor) allocation which convex combination gives $\mathbf{u}^{\cdot *}$. These transformed resource allocation for arbitrary n are: $u^*_i = (0, ..., 0, f_i(m), 0, ..., 0)$, $i = 1, 2, ..., n$, where $f_i(m)$ appears on the i-th position of the n-element vector.

This means that tasks are processed consecutively, each of them on m (all) available processors. □

It should be stressed that the above result *is valid for both: continuous and discrete functions*, describing the behavior of malleable tasks.

6 Example

We have two processors and a set of two tasks T_1, T_2 with processing demands $p_1 = 2$ and $p_2 = 3$ units, respectively, and convex function $f_i = x^2$, $i = 1, 2$.

The minimum schedule length $C^*_{max} = p_1/f(m) + p_2/f(m) = 2/4 + 3/4 = 5/4$. We know that $u^*_i = f(r^*_i) = p_i/C^*_{max}$. Thus, $r^*_i = f^{-1}(p_i/C^*_{max})$.

We have, $r^*_1 = \sqrt{2/\frac{5}{4}} \approx 1,2649$ and $r^*_2 = \sqrt{3/\frac{5}{4}} \approx 1,5492$. On the other hand, $\sum_{i=1}^{n} r^*_i$ differs from $m = 2$. This processors allocation is wrong, although the point ($u^*_1 = 1,6$; $u^*_2 = 2,4$) is the intersection point of the boundary of set *convU* and straight line $u_i = p_i/C_{max}$, but it isn't a transformed resource allocation. The same (minimal) solution gives convex combination of vectors: $u^*_1 = [4, 0]$, $u^*_2 = [0, 4]$. It means that the tasks are processed consecutively on 2 processors. Thus, from (1) we have: $t_1(2) = 0,5$, $t2(2) = 0,75$ and $C^*_{max} = t_1(2) + t_2(2) = 1,25$ (see Figure 6).

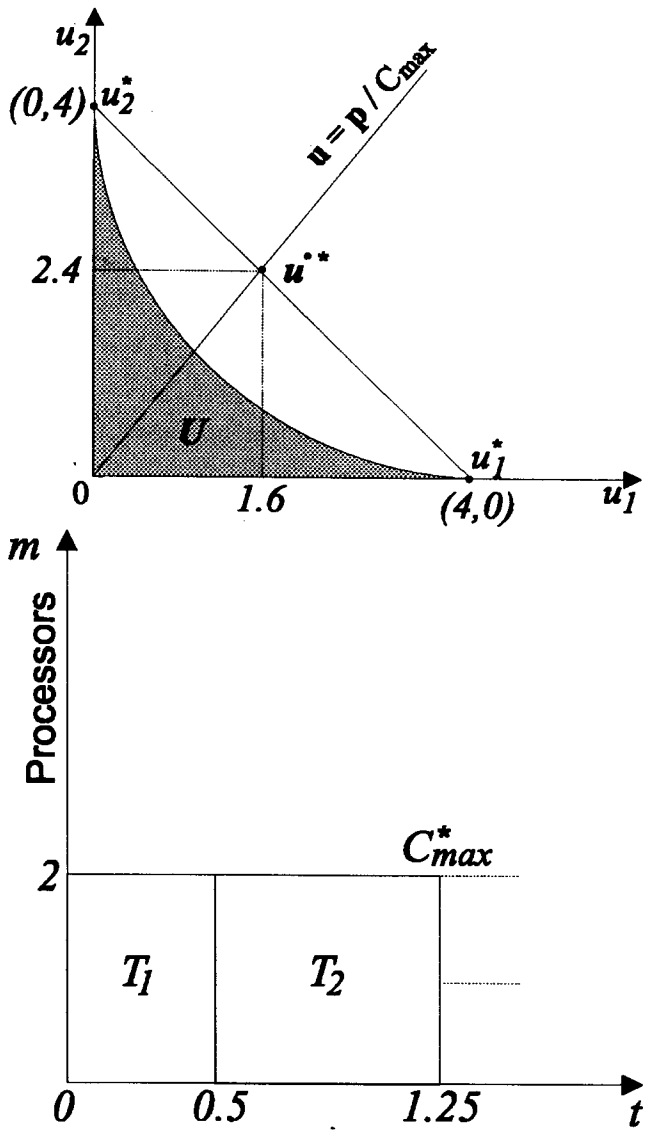


Figure 6: a) an optimal processors allocation, b) an optimal schedule.

7 Conclusion

In the paper, a concept of malleable tasks has been applied to an analysis of scheduling large computer programs for some specific applications. Using the results from control theory it has been proved that for convex functions relating speed of processing to a number of processors assigned to a task, gang scheduling strategy is optimal from the viewpoint of schedule length minimization. This is not the case of concave curves, where heuristic approaches need to be analyzed. The work on this topic is in progress.

References

[Barrett et al., 1993] **R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst,** *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1993. Obtainable at `http://www.netlib.org/templates`.

[Bernard, 1997] **P.E. Bernard,** Parallelisation et multiprogrammation pour une application irreguliere de dynamique moleculaire operationnelle, *Mathematiques appliques*, Institut National Polytechnique de Grenoble, 1997.

[Bernard et al., 1999] **P.E. Bernard, T. Gautier, D, Trystram,** Large scale simulation of parallel molecular dynamics, In *Proceedings of Second Merged Symposium IPPS/SPDP 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, San Juan, Puerto Rico, 1999.

[1986] **J. Błazewicz, M. Drabowski, J. Węglarz,** Scheduling multiprocessor tasks to minimize schedule length, *IEEE Transactions on Computers* **35**, 1986, 389–393.

[Dongarra et al., 1999] **J. Dongarra, L. Duff, D. Danny, C. Sorensen, H. van der Vorst,** *Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools)*, Society for Industrial & Applied Mathematics, 1999.

[Drozdowski, 1996] **M. Drozdowski,** Scheduling multiprocessor tasks - an overview, *European Journal of Operational Research* **94**, 1996, 215–230.

[Du and Leung, 1989] **J. Du, J.Y-T. Leung,** Complexity of scheduling parallel tasks systems. *SIAM Journal on Discrete Mathematics* **2**, 1989, 473–487.

[Feitelson and Rudolph, 1992] **D. Feitelson, L. Rudolph,** Gang scheduling performance benefits for fine-grain synchronization, *Journal of Parallel and Distributed Computing* **16**, 1992, 306–318.

[Golub and Van Loan, 1996] **G. Golub, C. Van Loan,** *Matrix Computations*, Johns Hopkins Studies in the Mathematical Sciences, The Johns Hopkins University Press, Baltimore, 1996.

[Gustafson, 1988] **J. L. Gustafson, Reevaluating Amdahl's law,** *Commun. of the ACM* **31**, 1988, 532–533.

[Lloyd, 1982] **E. Lloyd,** Critical path scheduling with resource and processor constraints, *Journal of the ACM* **29**, 1982, 781–811.

[Ludwig, 1995] **W. T. Ludwig,** *Algorithms for scheduling malleable and nonmalleable parallel tasks*, PhD thesis, University of Wisconsin - Madison, Department of Computer Science, 1995.

[1999] **G. Mounié, C. Rapine, D. Trystram,** Efficient approximation algorithms for scheduling malleable tasks, In *Eleventh ACM Symposium on Parallel Algorithms and Architectures (SPAA '99)*, ACM, 1999, 23–32.

[1982] **J. K. Ousterhout,** Scheduling techniques for concurrent systems, In *Proc. 3rd Int'l. Conf. on Distr. Computing Sys.*, 1982.

[1991] **G.N.S. Prasanna, B.R. Musicus,** The optimal control approach to generalized multiprocessor scheduling, *Algorythmica*, 1995.

[Rapine et al., 1998] **C. Rapine, I. Scherson, D. Trystram,** On-line scheduling of parallelizable jobs, *Lecture Notes in Computer Science* **1470**, 1998.

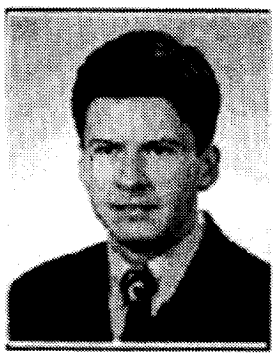
[Scherson et al., 1996] **I. Scherson, R. Subramanian, V. Reis, L. Campos,** Scheduling computationally intensive data parallel programs. In *Ecole francaise de parallelisme, reseaux et systemes. Placement dynamique et repartition de charge : application aux systemes paralleles et repartis*, Presqu'ile de Giens, France, INRIA, 1996, 107–129.

[Schwiegelshohn et al., 1998] **U. Schwiegelshohn, W. Ludwig, J. Wolf, J. Turek, P. Yu,** Smart SMART bounds for weighted response time scheduling, *SIAM Journal on Computing* **28**, 1999, 237–253.

[Turek et al., 1992] **J. Turek, J. Wolf, P. Yu,** Approximate algorithms for scheduling parallelizable tasks, In *4th Annual ACM Symposium on Parallel Algorithms and Architectures*, 1992, 323–332.

[Węglarz, 1982] **J. Węglarz,** Modelling and control of dynamic resource allocation project scheduling

Jacek Blazewicz *.Ph.D. - 1977, Habilitation - 1980 is a Professor of Computer Science, Deputy Director of the Institute of Computing of Science at the Poznañ University of Technology. He is a member of SIAM, Mathematical Programming Society, Polish Computer Science Society and a member of Editorial Boards od Parallel Computing, Journal of Scheduling, Journal of Heuristics, Mathematics of Industrial Systems. His area of interest includes: scheduling theory, theory of algorithms, parallel computing, computational biology. He has been also awarded Euro Gold Medal for his scientific achievements.*



Maciej Machowiak*, M.Sc - 1997 is a Ph.D. Student at the Institute of Computing of Science, Poznañ University of Technology. His area of interest includes: parallel computing, scheduling in parallel computer systems.*

Jan Weglarz*, Ph.D.1974, Habilitation - 1977. Is a Professor of Operational Management, Director of the Institute of Computing of Science at the Poznañ University of Technology, and also director of the Poznañ Supercomputing and Networking Center. He is full member of the Polish Academy of Sciences. His area of research covers a large variety of resource allocation models, algorithms and applications.*

Gregory Mounie *was PhD student at the LMC and ID laboratories of the IMAG institute of Grenoble from sept 1996 to sept 2000. Grants was provided by the French Ministry of Research and Joseph Fourier University of Grenoble. Main research interest was scheduling for hight performance parallel computing applications.*

Denis Trystram *was born in Paris, France, he received respectively two PhD from Institut National Polytechnique in Grenoblein Applied Mathematics in 1984 and in Theoretical Computer Science in 1988. He is leading the «Parallel Processing» group at LMC-IMAG. He is currently Regional Editor for Europe for the Parallel Computing Journal. He has published 5 books (one was translated in english), and edited 4 books, including the recent Handbook of Parallel and Distributed Processing by Springer Verlag. He has published more than 60 articles in international journals and as many international conferences.*