SCHEDULING DIVISIBLE TASKS UNDER PRODUCTION OR UTILIZATION CONSTRAINTS

By

Luis Fernando de la Torre Quintana

A thesis submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTING AND INFORMATION SCIENCES AND ENGINEERING

UNIVERSITY OF PUERTO RICO
MAYAGÜEZ CAMPUS

July, 2009

Approved by:

Domingo Rodríguez, Ph.D Member, Graduate Committee	Date
Kejie Lu, Ph.D Member, Graduate Committee	Date
Manuel Rodríguez, Ph.D Member, Graduate Committee	Date
Jaime Seguel, Ph.D President, Graduate Committee	Date
Anand D. Sharma, Ph.D Representative of Graduate Studies	Date
Néstor Rodríguez, Ph.D Program Coordinator	Date

Abstract of Dissertation Presented to the Graduate School
of the University of Puerto Rico in Partial Fulfillment of the
Requirements for the Degree of DOCTOR OF PHILOSOPHY

SCHEDULING DIVISIBLE TASKS UNDER PRODUCTION OR UTILIZATION CONSTRAINTS

By

Luis Fernando de la Torre Quintana

July 2009

Chair: Néstor Rodríguez
Major Department: Electrical and Computer Engineering

Several problems in science and engineering admit algorithmic solutions that demand a large amount of computing time. Among these applications are genotype sequencing, gene sequence comparison, protein folding, quantum chemistry, computational fluid dynamics, and Earth simulation. In most of these cases, a single computer does not provide enough computing power to satisfy these needs, and therefore, the design of parallel methods is of crucial importance. It has been observe in practice that many of these algorithmic solutions acquire the form of a master-worker algorithm. Due to their availability and low cost, heterogeneous networks of computers are becoming a popular alternative for these implementations. One problem, frequently faced by implementers is how to divide and distribute the parallel segments of computing tasks among the computers. This is the essence of the so-called task scheduling problem. Efficiently managing the computations is a difficult and challenging problem. This efficiency depends on the number of rounds of computation, the sizes of the data chunks sent in a round, and the number and the activation sequence of the participating workers.

In this dissertation variants and extensions of ideas related to the scheduling of master-worker tasks on heterogeneous star networks are introduced. Some of these ideas were previously discussed in the form of theoretical frameworks for steady-state scheduling or as a divisible load theory. This dissertation combines some elements of these previous works to construct a new framework, and from it, an efficient algorithm (SCOW) for identifying a deterministic scheduler for clusters of workers. SCOW produces the parameters of a periodic user-level scheduler for a single-program multiple-data implementation of a master-worker parallel solution. SCOW minimizes the job make-span under either maximal production per period, or perfect worker utilization. The efficiency of the scheduler identified by SCOW is demonstrated through comparison with other schedulers, including those derived from the above mentioned theoretical frameworks. As shown in the simulation an actual computer runs, the scheduler identified by SCOW outperform in most cases those produced by the previous frameworks.

Resumen de Disertación Presentado a Escuela Graduada
de la Universidad de Puerto Rico como requisito parcial de los
Requerimientos para el grado de Maestría en Ciencias

SCHEDULING DIVISIBLE TASKS UNDER PRODUCTION OR UTILIZATION CONSTRAINTS

Por

Luis Fernando de la Torre Quintana

Julio 2009

Consejero: Néstor Rodríguez
Departamento: Ingeniería Eléctrica y Computadoras

Muchos problemas en ciencias e ingeniería admiten soluciones algorítmicas que demandan una gran cantidad de tiempo de cómputo. Entre estas aplicaciones están la secuenciación de genes, la comparación de secuencias de genes, el plegamiento tridimensional de proteínas, problemas en química cuántica, dinámica de fluidos y simulación de fenómenos terrestres. En la mayoría de estos casos, un solo computador no proporciona suficiente poder de cómputo para satisfacer estas necesidades, y por lo tanto, el diseño de métodos paralelo es de crucial importancia. Se ha observado en la práctica que muchas de estas soluciones algorítmicas adquieren la forma de un algoritmo de tipo maestro y trabajador. Las redes heterogéneas de computadoras, debido a su disponibilidad y bajo costo, se están convirtiendo en una alternativa popular para estas aplicaciones. Uno de los problemas que a menudo enfrentan los programadores es cómo dividir y distribuir los segmentos de tareas de una computación paralela entre las computadoras. Esta es la esencia del llamado problema de planificación de tareas. La ejecución eficiente de los cómputos es un problema difícil. Esta eficiencia depende del número de rondas de cómputo, el

tamaño de los trozos de los datos enviados por rondas, y el número y la secuencia de activación de los trabajadores participantes.

En esta disertación se introducen variantes y extensiones de las ideas relacionadas con la planificacin de tareas del estilo maestro y trabajador en redes heterogéneas con forma de estrella. Algunas de estas ideas fueron previamente discutidas en la forma de marcos teóricos para planificaciones de tareas en sistemas en estado estable o como una teoría de división de cargas. Esta disertación combina algunos elementos de estos trabajos previos para construir un nuevo marco teórico, a partir del cual se construye un algoritmo eficiente (SCOW) que sirve para identificar un planificador de tareas determinístico para un grupo de trabajadores. SCOW produce los parámetros de un planificador periódico, a nivel de programador para programas escritos en estilo de un único programa pero con múltiples datos (SPMD) de algoritmos paralelos en el estilo de maestro y trabajador. SCOW identificar estos parámetros del planificador que minimiza el tiempo de ejecución del trabajo ya sea bajo la restricción de máxima producción por período o la de utilización perfecta de trabajadores. La eficacia del planificador identificado por SCOW se demuestra a través de la comparación con otros planificadores, incluidos aquellos derivados de los marcos teóricos mencionados anteriormente. Como se muestra en la simulación y en mediciones de aplicaciones reales, los planificadores identificados por SCOW superan en la mayoría de los casos a aquellos producidos por los anteriores marcos teóricos.

To my parents Maria and Rafael

and my love Oli

ACKNOWLEDGMENTS

My most sincere acknowledgments to my advisor Dr. Jaime Seguel for his tireless guidance and supervision, thanks for being like a father to me.

To the doctoral program in Computing and Information Sciences and Engineering (CISE) for your support.

Thanks to the other members in my graduate committee Drs. Domingo Rodríguez, Kejie Lu and Manuel Rodríguez for useful valuable comments.

Thanks to Dr Samuel P. Hernandez for his guidance, support and be a friend for me.

This work was supported in part by NIH grant MARC PAR-03-026.

Thanks to my wife Oliva for useful comments that greatly improved the readability of this dissertation.

At last, but the most important I would like to thank my family: Maria, Rafael, Julio, Hector and Pedro for their unconditional support and inspiration.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

MP	Maximal Production.
MU	Maximal Utilization.
MG	Mapping Group.
StarAffine	Star-network platform with Affine model.
StarLineal	Star-network platform with Linear model.
SPMD	Single-Program Multiple-Data style.
SCOW	Scheduler for Clusters Of Workers.
UMR	Uniform Multi-Round.
UMR2	Second version of UMR
MI	Multi-Installment.
SSS	Steady State Scheduling.
DLT	Divisible Load Theory.
LTDJ	Load- and Task-Divisible Jobs
SPMD	Single-Program Multiple-Data.
MPI	Message Passing Interface.
MMP-MP	Make-span Minimization Problem constrained to the Maximal Production.
PUMP	Perfect Utilization and Maximal Production.
MM-PWU	Make-span Minimization under Perfect Workers Utilization.

LIST OF SYMBOLS

X:	Total amount of core tasks to be processed for a given application.
p:	Total number of workers in a cluster platform.
x_i:	Number of tasks assigned to worker i.
n:	Integer number of round.
$\bar{x}$:	Array of worker chunk sizes $\bar{x} = (x_1, x_2, .., x_p)$
P_i:	Represents the ith worker.
P:	Set of workers.
e:	Master operation of packaging a group of tasks.
s:	Master operation sending a package of tasks.
l:	Transport operation of a package of tasks from master to worker.
r:	Worker operation of receiving a package of tasks.
w:	Worker operation of executing a group of tasks.
g_i, h_i:	Auxiliary Mapping for worker i.
OP:	Fixed latency for the respective *op* operation.
op:	The constant for the linear term the *op* operation.
q:	Identify the subgroup size of worker in the solution.
T:	The Time period of the periodic schedule
R:	Time spent by the master in a round of data retrieval and send operations
L:	Maximum time spent by a network link in transmitting their data packages in a round
C:	Maximum time spent by a worker in completing the reception of the data and execution of the corresponding agglomerated core tasks.
χ_i:	Is a Boolean variable equal 1, if the worker i is selected in the solution.
F:	Set of input mapping in a (T, p, F)-partition of X
`nRound`	Actual number of rounds
`nWorkers`	Actual number of participating workers
`nCoreTasks`	Actual number of core tasks.

CHAPTER 1
INTRODUCTION

Several science and engineering problems are computationally intensive. Among them are problems of contemporary interest such as genotype sequencing and gene sequence comparison, protein folding and gene network simulations [1–3], and others such as computational fluid dynamics and environmental simulations. In most of these cases, a single computer does not provide enough computing power to satisfy these needs, and therefore, the design of parallel methods is of crucial importance. Some of these applications admit computational solutions implementable over parallel platforms [4, 5]. Furthermore, some of these problems have certain level of load and task distribution which render naturally to master-worker implementations [6, 7]. Due to its availability and low cost both, homogeneous and heterogeneous networks of computers are increasingly becoming the alternative of choice for computational scientists. In [8] it is remarked that 82% of the registered parallel computing systems that make the cut for the top 500 performance competition are clusters. The pervasive presence of clusters brings about the need for the development of efficient methods for distributing the loads and tasks of a parallel job among the computing elements of these systems. The actual development of such methods, is nonetheless a challenging problem, and has remained unsolved in several instances for many decades. In theoretical terms, the search for these methods is referred as the *task scheduling problem*. Coffman [9] defines a schedule as follows: *"Given a set of tasks, a set of resources and a partial order representing the precedence relation between tasks; a schedule is a mapping of each of the tasks to a point in time and*

space, so that each processor executes only one task at a time, and the partial order is satisfied". By scheduling it is understood a set of mathematical techniques and criteria to identify the best scheduler under well-defined metrics and restrictions. Most scheduling metrics seek an efficient use of the time and the computing resources. Such a goal is often difficult to achieve. In particular, when it comes to scheduling tasks in a cluster, due to resource selection, the problems admits only combinatoric solutions [10]. Even when the cluster is configured as a simple tree or when the tasks are completely independent, the problem of finding a scheduler that minimizes of the total execution time remains NP-complete [10–14].

1.1 Basic Definitions and Concepts

This dissertation agrees with the common definitions of parallel and distributed computing under differences, as presented in [15]. These are:

Distributed computing is a method of solving computational problem by dividing the problem into many tasks run simultaneously on many hardware or software systems, which may be individual processing or storage elements, or programs, or individual computer systems, all running within a loosely to tightly controlled operating framework.

Parallel computing [is] the simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads.

As for the differences and similarities between parallel and distributed computing, the same source [15] comments: *In distributed computing, the individual tasks that a program is divided into, communicate usually over a computer network. Distributed computing is similar to parallel computing, but parallel computing is most commonly used to describe program parts running simultaneously on multiple processors in the same computer. Programs in distributed computing often must deal with heterogeneous computing environments, network links of varying latencies, and unpredictable failures in the network or the computers.*

Some current paradigm implementing parallel an distributed computing systems are: cluster computing, grid computing, and cloud computing. This dissertation concentrates on the cluster computing paradigm which is the oldest in the previous list. Cluster computing is still the paradigm of choice for the parallelization of scientific problems which is the ultimate aim of the scheduler developed in this dissertation.

This dissertation assumes compute systems that are ideally free of external perturbations. For example, it is assumed that the computer time is not shared by other users and that no interruptions in the availability of computer nodes occur. Therefore, problems such as random variations in compute time due to the sharing of computer resources or burst in communication demands, are not included in the model. In general, considerations such as random variations in processing time do not allow close solution to the scheduling problem, since they are modeled with random variables and stochastic processes. The model in this dissertation are more intended to capture steady state system behaviors. Nonetheless, the fact that the schedule derived from the theoretical framework presented in this work is identifiable in real time, allows for a quick reaction to changes in the computing environment. For example, if a computed node fails and no check-point/restart system is running, the schedule can be recomputed with the remaining compute nodes at a conveniently selected time in the execution of the job so that optimal make-span is achieved under the new circumstances.

The overall conception in this dissertation is that of a deterministically characterized system, this is a system whose evolution in time segment is completely determined by an observation of some of its key characteristics. Therefore, feedbacks into the system are not taken into consideration.

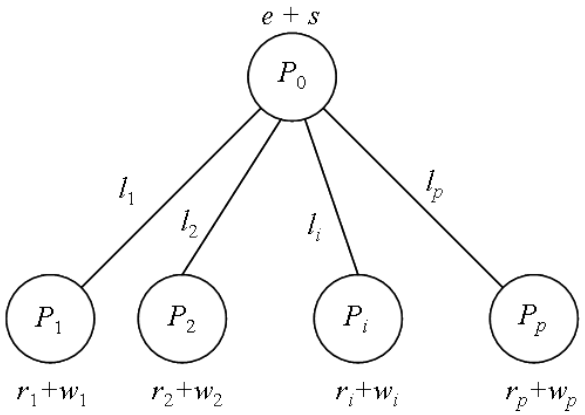

Figure 1–1: Heterogeneous Star Graph, with the Affine Cost Model

1.2 Target problems

This dissertation addresses some specific scheduling problems, which are summarized in this section. First of all, are the specific characteristics of the data and the job. This work concentrates on *load- and task-divisible jobs* (LTDJ). These are described in terms of X, which denotes a normally large number of indivisible tasks, called core tasks. In most practical situations, these X core tasks are indeed, a single core task repeated over a large numbers of different data chunks. Thus, in most practical situations, the problem falls naturally under a master-worker paradigm where a master processor distributes data chunks across the workers, which in turn, execute the core tasks concurrently. One such distributions is called *round of data installments*.

As an example, let X=1000 be a given number of core tasks. If there are p=16 workers available and a decision is made to assign 20 core tasks to each of 10 selected worker, then, there will be

$$\frac{1000}{20 \times 10} = 5 \tag{1.1}$$

rounds of execution of the 20 core tasks on each selected workers. This particular distribution of job is executed by a single program multiple data (SPMD) pseudo code like the following:

```
For j = 1, 5
    If Master process
      For i = 1, 10
        Prepare data for 20 tasks to be sent to worker i
        Send the prepared data to worker i
    Else
      Receive the data sent by the master,
        For j = 1, 20 //may be different for each worker
          Compute task
```

these parameters were chosen arbitrarily. There is no guaranty that the performance of the 10 workers will be the optimal. Indeed, the performance will vary with the number of tasks selected. Finding the number of tasks that produces the optimal schedule is the central problem addressed in this dissertation. Finding the optimal solution for this problem depends on several factors. Among them are the time spent in transmitting and computing the data, the choice of workers, and the number of round. Another important consideration is the architectural framework on which the master-worker paradigm is implemented.

In this dissertation, the platform is assumed to configure as a star topology, which is commonly denoted in the literature as StarAffine. The StarAffine network is characterized by the assumption that the time costs of all operations is an affine mapping in the amount of core tasks. Figure 1–1 is an illustration of a StarAffine network. In this, the root node represents the master and the leaves, the workers. The edges, in turn, represent the communication links. In this dissertation it is assumed that the master can communicate with a single worker at a given time, and the communications may overlap computations. This is usually referred as *full overlap, single-port model*.

Finally, and mainly because of its easy of implementation, this dissertation concentrates on periodic schedulers. As a matter of fact, the resulting scheduler will have three phases. An initial phase where the master sends the first data chunks to each processor. Next the schedule enters a truly periodic phase characterized by rounds of data installments of equal size. This phase dominates the schedule. Throughout the periodic phase, the compute system is expected to run with a maximal efficiency, executing the maximal possible number of core tasks per period. The final phase consists simply in the execution of the computations that remains after the periodic phase is finished.

The main objective of any scheduler is minimizing the make-span; this is the time difference between the start and finish of a sequence of jobs. Out of the three previous phases, the second phase dominates in the make-span. This dissertation presents a novel solution to the make-span minimization problem, based on restrictions over the periods of the schedule. Such restrictions can be either maximal production, which is the maximization the number of tasks completed within a period; or maximal utilization, which corresponds to the choice of resources in such way that the total idle time is minimal.

These restriction are imposed on a mathematical model of the system, which is based on some basic performance measurements. This leads to two optimization problems, namely the maximal production problem and the maximal utilization problem. Each of these problems is solved keeping the period as a variable. Each of these solutions are used in turn, as restrictions to make-span minimization problem. The solution to the latter minimization problems has the additional advantage of optimizing the use of the computing resources.

The above schema use the number of core tasks X and the basic measure of system to return the optimal values to number of tasks per worker, number of rounds and the selected workers. This parameter yields in a user-level scheduler that

is particularly appropriate for scheduling tasks programmed in a SPMD style. In SPMD rounds are controlled by an external do-loop, which imposes the mentioned periodic character in the execution of the job. The general aspect of such an SPMD code is

```
For j = 1, nRounds
    If Master process
      For i = 1, nWorkers
        Envelop agglomerated data chunks for worker i
        Send agglomerated data chunk to worker i
    Else
      Receive agglomerated data chunk,
        For j = 1, nCoreTasks[Id worker] //may be different for each worker
          Compute core task
```

where `nRound, nWorkers` and `nCoreTasks[i]` are the number of rounds, the number of participating workers and number of core tasks to each worker respectively.

1.3 Previous scheduling methods

Several mathematical frameworks for scheduling divisible tasks has been developed. However, none of them renders a user-level scheduler directly. This dissertation takes some fundamental elements from these framework to develop a user-level scheduler and as a base for comparison.

Among the frameworks is the one proposed in [16, 17]. The main idea in [16] is to relax the problem's objective function by maximizing the throughput when the system operates in steady-state. In [16] it is proved that such relaxation leads to a scheduler that is asymptotically optimal. This proof agrees with intuition in the sense that if the total amount of load is large, initialization and clean-up phases may be ignored. Under this circumstances, minimizing the total make-span is almost

equivalent to maximizing the throughput of the whole system. The framework proposed in [16] finds the best scheduler in $O(p\ log(p))$ time, where p is the number of workers.

Another framework is discussed in [17]. This framework extends the work of [16] to a system modeled by affine mappings referred above as StarAffine. This extension uses the solution in [16], which is based in a linear model, but imposes a start times on each mapping when building the scheduler. As a result, the scheduler may present idle time gaps at run time. A drawback in this framework is its inability to provide the optimal number of rounds. Indeed the whole framework is base upon the premise that the scheduler is independent of the total amount of work, and therefore, the optimal number of rounds is not part of the theory. In practice, this theoretical gap can only be filled heuristically.

Another scheduling framework, which is called uniform multi-round algorithm (UMR) is introduced in [18]. UMR is uniform in the sense that it contemplates either uniform data chunk sizes per round, or uniform worker compute time, per round. The scheduler is designed to absorb the startup time of the send operations by increasing the data size or the time length in each round. The uniformity also allow for the determination of the necessary conditions for a full platform utilization and the optimal number of round. In [19] the concept for UMR is extended to what is called UMR2. This extension includes both, increasing and decreasing data chunk sizes or compute time lengths, per round. This provides UMR2 with the ability to use more resources that UMR and adds to the scheduler, a resource selection policy.

A survey of divisible load scheduling frameworks is presented in [20], including among others, the work in [17] and the UMR technique. Finally, [21] presents an estimation of the optimal number of rounds for the scheduler proposed in [17]. This estimation is based on an upper bound for the throughput of the rounds of the scheduler.

1.4 Approach and goals

This dissertation develops an user level periodic scheduler for clusters of workers (SCOW) that tunes the parameters `nRound, nWorkers` and `nCoreTasks` of a SPMD implementation of a master-worker parallel solution. SCOW minimizes the job make-span under either maximal production per period, or perfect worker utilization. Under the constrain of maximal production per period, the scheduler developed is similar in spirit to the one developed in [21], but it differs in taking into account the length of the period and the StarAffine network model. The resulting scheduler provides an optimal throughput for homogeneous networks and a good approximation to the optimal throughput for heterogeneous networks. This places the SCOW method as a sort of middle point between UMR and UMR2, in the sense that it absorbs the cost of the send operations maintaining a constant period T through the rounds of data installments. Unlike [21], the scheduler produced by SCOW gives the possibility of finding an exact subset of worker over which the system runs with perfect worker utilization. In this way, SCOW eliminates idle gaps in the worker's computation. A last round modification is also introduced, to ensure that all the workers end operating at the same time.

Since SCOW is intended to be used in real life applications, special care is taken in the validity of the model. Through experimentations, it has been observed that the model is accurate within intervals in the numbers of core tasks. These intervals are called Affine Windows in this dissertation. The effects of Affine Windows is incorporated in the scheduler in a postmortem fashion. When this is done, the set of workers necessary for full system utilization may be reduced.

The main contribution of this dissertation are:

Realistic model Unlike previous works whose basic models are focused in particular aspects of the targeted system, the basic mathematical model used in this dissertation takes into account all the operations that can possible occur in the

execution of the master-worker SPMD program in a cluster. This mathematical model is also realistic in the sense that execution time of all the operations include both, startup and the execution times. Thus, execution times are presented as affine mappings on the number of agglomerated tasks. The constant part of the mapping corresponds to the operation's overhead. These mappings are obtained by regression over a sample of measurements of the system's behavior. Furthermore, the validity of these affine representations is taken in to account when identifying the schedule.

Maximal production Previous works have addressed the problem of obtaining a maximal throughput under fixed task sizes and using a linear mapping instead of an affine one for representing the time execution of the operations. In this dissertation the problem is cast in terms of affine mappings, and variable tasks sizes. In doing this, the concept of maximal throughput is replaced with that of maximal production, which is similar in spirit but more amenable for mathematical treatment.

Maximal Utilization This dissertation introduces a new metric for assessing the use of the cluster resources in the solution of a problem. The throughout revision in the literature that was performed for this dissertation showed that this metric has not been used before in the context of scheduling. Nonetheless, maximal utilization is an important attribute in the efficiency of a scheduler, specially in the cases where large cluster systems are available for an equally large number of users.

Resource selection Solving the resource selection problem is one of the most difficult steps in the design of a schedule. This dissertation presents two mathematically well-founded solutions to this problem. First, is a result that finds the best subset of workers for achieving maximal production per period. Second, is the result that allows the identification of task sizes and number of workers that achieve the perfect worker utilization, in the sense that all the selected workers operate continuously throughout the execution of the job.

Maximal production and perfect utilization of the workers Another important contribution of this dissertation is the theoretical result that establishes that maximal productions and perfect workers utilization are achieved only in a finite number of values of the variable T representing the period. Furthermore, such values are shown to play a crucial role in the partition of the domain of the period. The intervals in this partition are characterized by the fact that the optimizations problems of maximal production and perfect utilization in a period can be formulated with a fix number of workers on each period. This number of workers is given by the underlying mathematics of the construction of the partition. This theoretical result is essential for the solvability of the make-span minimization problem. The strategy pursued in this dissertation consists in solving each of the finite optimization problems over the intervals in the partition, this is either the maximal production or perfect workers utilization in a period; and then searching for the solution that has minimal make-span.

Last round modification Although frequently mentioned as a measure of load balance, not all theoretical frameworks for scheduling whit maximal throughput per round include explicitly the condition that all the workers end operating at the same time. In this dissertation an explicit method for achieving this aim, called last round modification method, is presented both, in theory and implementation. Last round modification is proved to improve significantly the efficiency the schedule whenever the period is large.

Practical implementation Unlike many of the previous theoretical scheduling frameworks the scheduler identified with the theoretical frameworks presented in this dissertation has been tested not only in simulation but also in actual problem solving situations. The result of real jobs confirm the theoretical and simulated superiority of the schedule built with the framework discussed in this dissertation.

SCOW is compared with the above discussed and other scheduling methodologies both, through numerical simulations and actual applications. The simulations show that the scheduler produced by SCOW renders better performance for LTDJ applications than job scheduled with the frameworks in [18, 19, 21]. Runs comparing a SCOW scheduled C++/MPI bioinformatics solver against a first-in, first-out scheduled, and a UMR scheduled implementation, also show the superiority of SCOW.

The rest of this dissertation is organized as follows: Chapter 2 is a revision of the literature on divisible task and steady-state scheduling theories. Chapter 3 introduces the main concepts used in this dissertation and examines in further detail, the frameworks introduced in [16, 18, 19, 21]. Chapter 4 discuses the maximal production and maximal utilization problem, and develops an optimal solution for homogeneous clusters together with an approximate solution for heterogeneous clusters. Chapter 5 introduces a recursion formula for the modification of the last round, and the theory behind the make-span optimization for both, maximal production and maximal utilization constrains. Chapter 6 presents numerically simulated and actual implementation comparisons of SCOW with other schedulers.

CHAPTER 2
LITERATURE REVIEW

Scheduling tasks in multiprocessor platform is a main problem in combinatorics [10]. There are two main aspects of the problem, each with a different degree of difficulty: scheduling over homogeneous networks and scheduling over heterogeneous networks. For some extent works area, these dependent directly to the relationship between the processors: homogeneous or heterogeneous system. For this problem has been demonstrated that the minimal makespan is NP-complete for the general cases [10–13], therefore when the platform is a simple tree [14] or the task are independents. The advantages that can be provided for a heterogeneous distributed system have attached the increment in complexity solution. The principal idea designs to work in a simple processor, makespan optimization, are poor utility in this system. The more popular alternative is approximation schedule, in [22] is proposed an interesting approximation algorithm based in steady-state optimal flow of packet, this algorithm produces an asymptotical optimal schedule when the total number of jogs tends to infinity. In [23] is first propose a solution to star network, this solution is achieved including additional constrains in the problem, "send works in no decreasing communication link order". After in [24] is presented a polynomial algorithms to distributed back propagation neural networks on heterogeneous networks of workstations, this problem is implemented with the master-worker paradigm without any initial communication cost. Some of these ideas are used in [25] to develop a polynomial time schedule for master-worker intended to optimize the throughput

of the system when operating at steady state. In more general problems, all these ideas are used in scheduling for divisible load application [20, 21, 23, 26].

Two approaches dominate among the methodologies developed for scheduling of master-workers tasks. These are: steady state scheduling (SSS)[16, 27] and divisible load theory (DLT) [20, 23].

In[23], a book that presents a comprehensive theory of divisible load scheduling The other basic reference for this work is the bandwidth-centric strategy for scheduling a set of equal-sized tasks over a heterogeneous, tree-shaped platforms [16, 27]. Both approaches lead to approximately optimal schedules. And both approaches have important drawbacks that limit their applicability in practice.

In [16], is presented an aim to maximizing the throughput of the system, an algorithm for finding the optimal number and sequential order of the processors, and the fraction of time that each processor must spend computing over a standard period of time, is discussed thoroughly. The resulting theoretical framework has the limitation of assuming linear communication and computation behaviors. Also, the model is not detailed enough to allow for any practical tests and implementations. Nonetheless, this work set the bases for the combine throughput-makespan optimization model proposed in this disertation.

2.1 Steady state scheduling

This first approach maximizes the network throughput by assuming that the main task is divided into independent equal-sized subtasks. The SSS methodology finds the optimal number of workers for maximizing the throughput [27], assuming that all parallel tasks have the same size which is kept fixed during the scheduling is indeed a big limitation. In most practical situations the size of the subtasks is a central factor in scheduling optimization, since it provides a way for adjusting the computations-to-communications ratio to the performance characteristics of the system. In [16, 27] a polynomial time algorithm for the periodic scheduling

of master-worker tasks with an optimal throughput over a system operating under steady-state, is presented. The computer network topology in these works is to be a tree, so the optimal throughput can be characterized using a bottom-up recursive equation traversing the tree. In this algorithm tasks have a fixed, uniform size, and communications and computations are modeled linearly. These assumptions are not realistic. Besides the possibility of incurring in large periods of different data installments, which complicates programming, the method may result in an overuse of data buffers., Nonetheless, the theoretical framework behind this method is highly elaborated and complete, bringing up a good understanding of the method for concept finding a the optimal number of processors for maximizing the throughput of a tree-shaped platform, operating in steady-state. The problem with the memory buffers is addressed in [26]. This work reviews the algorithm and analyzes the impact of the buffer sizes in the achievement of an optimal performance.

2.2 Divisible load theory

The Divisible load theory is used to model parallel jobs in which each subtask can be processed in parallel, and on any number of compute nodes [23]. This model finds applications in many sciences and engineering problems. In practice, divisible load applications are used in jobs that consist of low-granularity computations such as those that occur in image processing, volume rendering, data mining, and some bioinformatics applications [28, 29]. In the research of divisible workload scheduling, a critical topic of interest is the overhead incurred in the input/output data transfer time, to or from the master, and the latency incurred before starting the computations [20]. This second approach (DLT), divides the workload in a arbitrary number of tasks, and distributes them among the processors under the condition that these processors end operating at the same time. Although DLT allows the scheduler to adjust the subtask sizes, it introduces some problems of its own. In order to optimize the scheduling of tasks in DLT two methods are considered [23]: i) divide the

workload in big chunks, this in general reduce the overhead by constant latencies, *ii*) divide the workload in small chunks in multi-installments schedule, this enhancement the communication and computation overlap. A drawback in the divisible load schedule is that the optimal number of installments is in general, unknown and remains an open problem. In [30, 31] are the first research about this problem and in [29] is demonstrated that the optimal installments number to the lineal model is infinity.

For instance, in the initial versions of the DLT methodology used a single round of distribution, also called data installment, at the beginning of the operation [23]. Simulations showed that this form of the method was prone to introducing large latencies in a significant number of cases [23]. A modified version was later proposed to diminish these latencies. In the modified version, several rounds of data installments, each delivering data chunks of smaller sizes was propose as a way of keeping processing nodes as busy as possible. This modification, however, brought a new problem. It turned out that no polynomial time algorithm is known for computing the optimal number of such rounds of installments when the model is affine. As a consequence, one of the most influential parameters in the schedule cannot be efficiently determined. The DLT approach also presents other known problem the sequencing problem. This is the problem of determining the sequential order in which the load is assigned to the processors. It has been shown [26] that for an infinite number of tasks, the optimal order is the non decreasing order of the master to worker link capacities. The problem is still open for a finite number of tasks. Many works have been carried out where is supposed lineal and affine model. In [26] is deviced an unified discussion of divisible load scheduling results in star and tree networks. This Work has three main aspects: selection and order of the workers, computation of the chunk sizes at each round on multi-round algorithms and optimal number of rounds.

2.3 Single-round algorithm

One-Round algorithms is the ones of most studied topic in DLT. In this schedule, to each worker is assigned only a piece of load in the entirely schedule. Ones of the firs works in this topic is the Linear network [32, 33], the second ones is StartLineal [34, 35], a the last StartAffine model [36, 37]. A other related study in single round with memory constraints [26, 38, 39], that is a NP-complete problem. [26] gives a good summary of the computational complexities of Single-Round and Multi-round algorithms, for star/trees with linear cost model, the problem can be solved in polynomial, but for the mos general affine cost model, the complexity is unknown. In [38] a Mixed Linear Programming with possible exponential cost solution is proposed to solve the most general formulation of Single-round problem, But due this technique not has the possibility to overlap communication and computation, their performance is poor.

2.4 Multi-installment algorithm

The affine model is more realistic but the premise above (infinity installments number) is not true in this case. In [40], the multi-installment concept is extend to the use of affine model communication and computation models, and it is prove that an infinity number of installments is not optimal solution in this case. There are many algorithms based on the multi-installment paradigm. Among them are the Multi-Installment (MI) algorithm [30, 31], this MI assumes a purely linear time cost model. Other work in this area is Uniform Multi-Round (UMR) algorithms [18, 26, 41], which is a multi-round algorithm that proposes an optimal number of installments constrained to uniform restriction conditions.

The UMR algorithm is, among the Divisible Task Scheduling methods, the one that resembles most the SCOW methodology. The UMR method in order to determine the optimal number of rounds of data installments imposes a restriction on the size of the tasks: The sizes of the tasks sent during a round must be the same

if the network is homogeneous, and, if the network is heterogeneous, the time that each processor spends computing must also be the same. Without these restrictions is not possible to determine an optimal number of rounds of installments. Such optimal number of rounds appears as an open problem.

Models including data returns from the workers to the master before or after a parallel computation has also been studied. In [26], the Master/Slave paradigm is revised with this addition in mind. There, a polynomial algorithm is introduced for distributing a group of tasks over a start network within a fixed period of time. The problem of scheduling a group of tasks over a star network including communications before and after the execution of the tasks is proven to be NP-complete [26]. A heuristic method is presented for computing an approximate solution, as well.

Finally, both methodologies SSS and DLT are based on mathematical models of the underlying computing systems that are oblivious to some of its subtle, but nonetheless influential aspects. In particular, neither SSS nor DLT models depict a clear decoupling between send, transmission and reception operations, nor they include data buffers. In addition, the DLT model assumes the unrealistic property that a data load can be continuously divided. In the previously reviewed publications the central concepts and main theoretical elements of Divisible Tasks Scheduling are introduced. Among them are the elements used in this dissertation for minimizing the makespan aspect of the makespan-throughput duality.

CHAPTER 3
FOUNDATIONS

As stated in chapter 1, LTDJ assumes a large number X of core tasks. These core task can be agglomerated to produce different job sizes. These jobs are independent in the sense that neither ordering between them, nor synchronization among them is necessary.

3.1 Model and notations

As illustrated in Figure 1–1, the StarAffine network consists of $p+1$ processor, P= $\{P_0, P_1, P_2, ..., P_p\}$. The master processor is denoted P_0 while the p workers are labeled P_i, $1 \leq i \leq p$. There are p communication links from the master P_0 to each one of the workers P_i. Let x_i be the number of units of data load sent to worker P_i. The master performs two operations. The first operation is a message preparation that includes data retrieval and message pre-processing. This operation, which referred as envelop, is performed in $e(x)$ time units where x is a number of core tasks. The second operation is the sending of the message, called send operation, which is performed in $s(x)$ time units. The transmission of the messages through the communication links is in turn, modeled by mappings $l_i(x)$, $1 \leq i \leq p$. each of which measures time units that takes for a load x to be moved from the master to the ith worker. Each worker i performs two operations, as well. These operations are message reception, referred as receive; and the actual execution of the job, referred as computation. The worker takes $r_i(x)$ time units in performing a receive operation for the load of corresponding to x core tasks, and $w_i(x)$ time units in executing x core tasks. Without loss of generality it is assumed that the

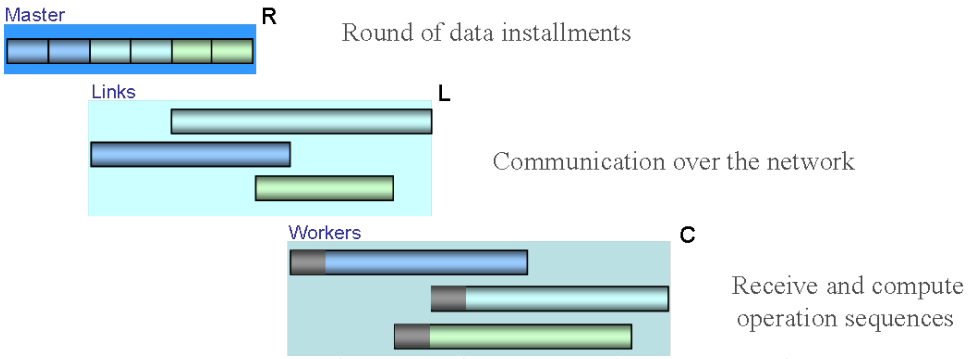

Figure 3–1: Theoretical Framework in a Period T

master does not execute any core tasks. Throughout this dissertation x_i represents the number of core tasks sent to worker P_i.

The different scenarios for the operation of the processors are surveyed in appendix B. As stated earlier, this dissertation assumes the *full overlap, single-port* model. In this model, the master can perform an envelop and send operations concurrently with transmissions over the communication links. Similarly, a worker can operate concurrently with transmissions over its own communication link with the master. This scenario is realizable in parallel systems whose compute elements are equipped with a front-end processors [23]. Some restrictions on this model yield important variants. Among them are *bus networks* and *homogeneous cluster*. A *bus networks*is a StarAffine network in which all the communication links are equal, in the sense that $l_i = l$ for all mappings. A *homogeneous cluster*, in turn, is a bus network in which all workers are equal. This is, the mappings $r_i = r$ and the mappings $w_i = w$ for all workers. In all the above cases, l, r and w are average mappings that are defined later in the Definition 6.

In general terms, the system operates as follows: within a round of data installments the master performs a sequence of data envelopes and send operations. Each data envelop and send operation is followed by a data transmission through the corresponding link. Receive and compute operations are performed by the workers upon the arrival of the data package. Envelop and send operations are performed

sequentially in the master, and receive and compute operations are performed sequentiality by the workers. Envelop and send operations in turn, are performed concurrently with data transmissions; while receive and compute operations are concurrently with data transmissions, as well. As a result, three major concurrent time segments are distinguished within a round. Figure 3–1 depicts these segments, which are called *master*, *links* and *workers*. These segments in turn, give rise to the quantities R, L and C described in Table 3–1.

Table 3–1: Theoretical Frame Work Description

Notation	Description
R	Time spent by the master in a round of data envelop and send operations
L	Maximum time spent by a network link in transmitting its assigned data packages
C	Maximum time spent by a worker in completing the reception of the data and execution of the corresponding agglomerated tasks.

Intuitively, a good scheduler maximizes the overlapping of these segments and minimizes the idle times between rounds.

3.1.1 Periods and continuity between periods

Figures 3–2 and 3–3 are Gantt Charts of periodic schedules with four workers and one master. Figure 3–3 depicts a schedule that saturates the communication network in the sense that there are no idle communication times, while Figure 3–2, depicts a schedule that has no gaps in the master's execution times. Both figures keep all except one of the workers operating continuously, as well.

Definition 1. *The period of the schedule, denoted by T, is defined to be the maximum between R, L and C.*

If $T = R$, the master process works continuously as depicted in Figure 3–2.

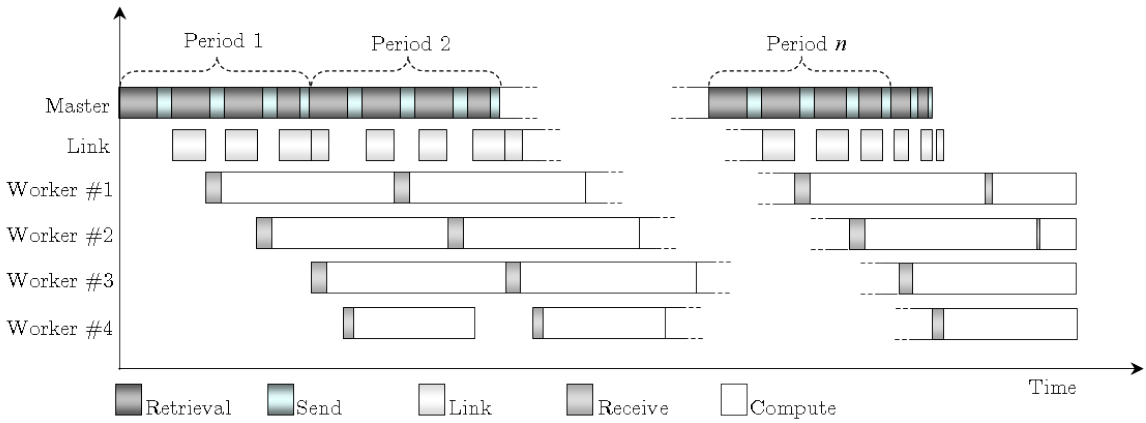

Figure 3–2: Continuous Master Utilization

Definition 2. *If $T = L$ or $T = C$, there is either:*

(a) *weak continuity, if at least one link or one processor operates with idle times between periods; or* (3.1)

(b) *strong continuity, if all links or all workers operate with no idle times between periods.* (3.2)

The parts in the above definition are not mutually excluding in the sense that it is possible to have strong continuity in the links and weak continuity in the workers, or vice verse.

Definition 3. *A schedule satisfying*

(a) $T = R = C$, $L \leq T$ *and* (3.3)

(b) *Workers operate in strongly continuous mode,* (3.4)

is said to be a balanced schedule.

When $L = T$, the schedule is said to be *perfect* balanced. By allowing the master to participate in the execution of core tasks, and if the schedule is such that $L = C = T$, then it is possible get $R = T$. This means that the master is saturated as well. A balanced schedule ensures that workers receive just as much work as they can concurrently do within T units of time. In section 4.3 it is shown that for each

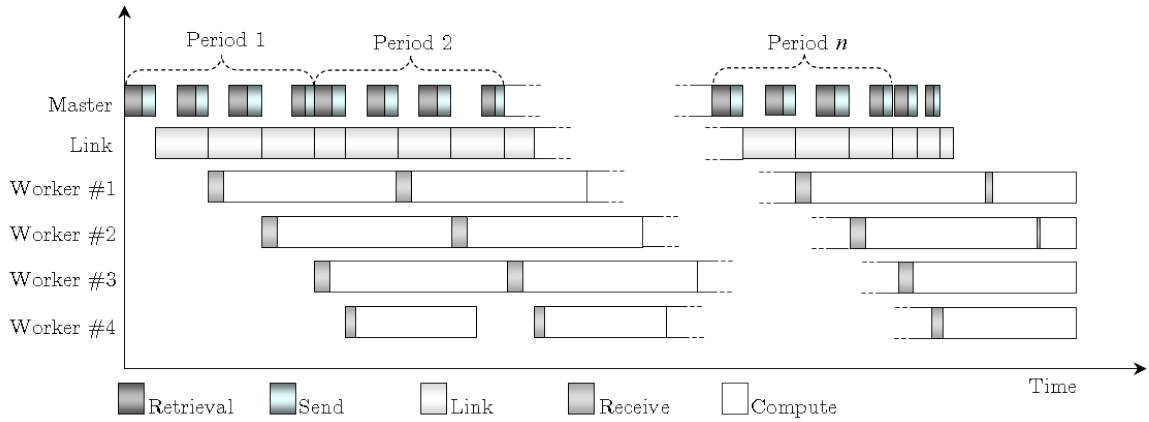

Figure 3–3: Continuous Network Utilization

choice of T there is either a balanced schedule or a best approximation to a balanced schedule. But, neither schedule can guarantee a minimal make-span.

3.1.2 The affine mappings

This subsection is a brief discussion of the affine maps in which the mathematical model is based. The model assumes that the execution times of each of the operations of data envelop, send, communication, receive, and execute tasks vary as an affine mapping on the number of agglomerated core tasks. This is, if *op* denotes anyone of these operations and x is a variable taking numbers of agglomerated core tasks,

$$op(x) = x.op + OP;\ op = e,\ s,\ l_i,\ r_i, \text{ or } w_i. \tag{3.5}$$

In 3.5 *op* is overloaded since it is use both, for the name of the mapping and for the scalar that corresponds to execution time of a single core task. This overloading is intended to keep consistency with notations used in references such as [20]. Yet another convention is the use of capital OP to denote the startup time of operation *op*. Thus, the mappings consist two constants, namely OP and *op*, and a variable x representing amounts of core tasks. The constants OP and *op* are use later as inputs for the algorithms and methods that conform SCOW. In practice, these constants are obtained by linear regression over a small set of measurements. Table 3–2 is a complete description for each of these mappings. It is worth remarking that

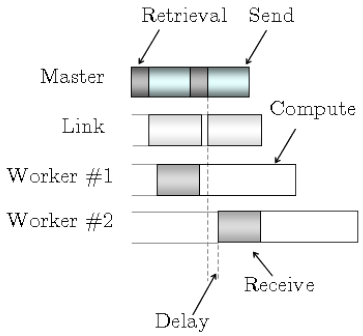

Figure 3–4: Observation of a Delay in a Real Measurement

assigning $x = 0$ tasks to a worker means that such worker does not participates the computation. When this happens, the resource selection problem appears as an important factor in the search for the best schedule.

For the purpose of analysis, the number of agglomerated tasks in (3.5) is supposed to be a nonnegative real variable $x \geq 1$. This is an unrealistic assumption, since agglomerated tasks yield natural numbers. A post-optimization discretization method, which is discussed in section 5.3, is applied to obtain an approximation to the optimal SPMD parameters.

Table 3–2: Mapping Description

Mapping	Description
$e(x)$	Time spent by the master in retrieving data and preparing a package for submission to a worker
$s(x)$	Time spent by the master in performing the send operation for sending a package to a worker
$l_i(x)$	Time spent in transporting data over the master- processor i link
$r_i(x)$	Time spent by processor i in performing the receive operation of a package
$w_i(x)$	Time spent by processor i in executing X atomic tasks.

The mapping $l_i(x)$ is often hard to measure directly. Its affine behavior is postulated from measurements performed with combinations of blocking and nonblocking send and receive operations for x agglomerated tasks. With these measurements is possible to model a system with or without front-end. Without front-end $s+l_i+r_i$ is

replace by $delay+r_i$, where $delay$ is the time between the start of the a blocking send operation and the start of the blocking receive operation, as shown in Figure 3–4. The next function expresses the cases of an observable, and that of a non-observable $l_i(x)$, in a single formula.

$$g_i(x) = \begin{cases} (s + l_i)(x), \text{ if communication and computation are concurrent} \\ delay(x), \text{ otherwise} \end{cases} \tag{3.6}$$

The notion of asymptotic domination plays an important role in the theoretical formulation of SCOW.

Definition 4. *Let f and g be real-valued mappings with values in the real numbers. Then f is said to dominate g asymptotically if there exists a real number $x_\epsilon \geq 0$ such that:*

$$g(x) \leq f(x), \ for\ all\ x \geq x_\epsilon. \tag{3.7}$$

In this work, h_i denotes the mapping that dominates asymptotically between the mappings $e + s$ and l_i. This is,

$$h_i(x) = \begin{cases} (e + s)(x), \text{ if } (e + s) \text{ dominates over } l_i; \text{ or} \\ l_i(x), \text{ if } l_i \text{ dominates over } (e + s). \end{cases} \tag{3.8}$$

In general, if h_i dominates over $(r_i + w_i)$, the parallelization is deemed poor A.2. Consequently, in order to develop a consistent scheduler, it is assumed in this dissertation that r_i+w_i dominates over h_i for at least a few workers $(1 \leq i \leq p)$. The mappings in Table 3–2 and the auxiliary mapping are collectively called Mapping Group (MG).

3.1.3 Affine windows

The regressions in Figure 3–5 show that the affine mappings in MG adjust to the data with minimal errors, within intervals of the values of x core tasks, referred before as Affine Windows. Outside the Affine Window, the model loses its

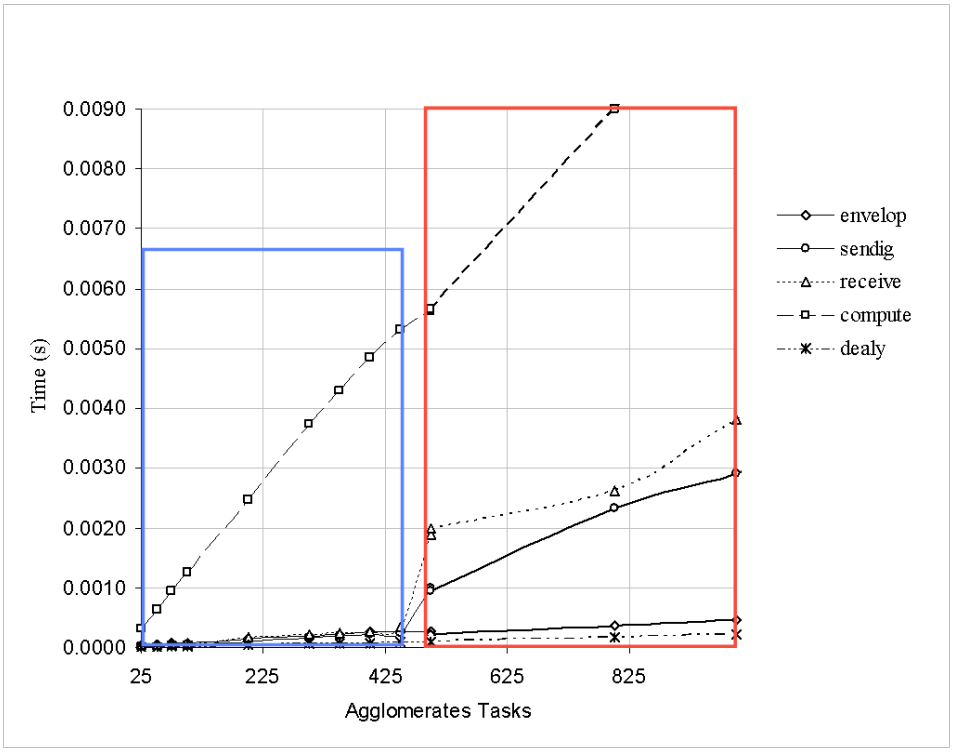

Figure 3–5: Affine Windows Example

validity. Figure 3–5 depicts actual homogeneous system measurements performed using blocking send a receive operation in a MPI environment. The first Affine Window is completely determined by the experiment and correspond to the values of agglomerated core tasks within the interval [25,450]. The second Affine Window is not completely determined as experiment where stopped at $x = 2500$ agglomerated tasks. The abrupt changes in the slope of the mappings between the windows is most probably due to changes in the policies in the MPI implementations of the send and receive operations [42].

3.1.4 Homogeneous clusters and average mappings

The concept of homogeneous cluster has not been formally defined. Informally, by an homogeneous cluster it is understood a cluster formed by machines of the same hardware characteristics and similar communication links. However, similar hardware characteristics do not always guarantee an homogeneous behavior. In this dissertations an homogeneous cluster is a system that satisfies the next definition.

Definition 5. *Let p be the number of workers available to the master and $TOL > 0$ be a real number. A cluster is said to be homogeneous within a tolerance TOL, or TOL-homogeneous if for all i, j, $1 \leq i < j \leq p$ and for each concurrent operation op,*

$$|op_i - op_j| \leq TOL \text{ and } |OP_i - OP_j| \leq TOL. \tag{3.9}$$

A system is said to be heterogeneous with respect to a tolerance TOL, or TOL-heterogeneous, if there are i, j, $1 \leq i < j \leq p$, and an concurrent operation op such that

$$|op_i - op_j| > TOL \text{ or } |OP_i - OP_j| > TOL. \tag{3.10}$$

Definition 5 is based solely on time measurements. TOL is intended to be an upper bound for uncertainties in the measurements. If this is the case, TOL is in fact, an error bound for

$$Err(op, i, j, x) = |x.op + OP_i - (x.op_j + OP_j)|. \tag{3.11}$$

in the sense that

$$Err(op, i, j, x) \leq (x + 1) \times TOL. \tag{3.12}$$

If a system is TOL-homogeneous or simply, homogeneous; the affine mappings $i = 1, ..., p$ associated with a concurrent operation can be replaced with their average mappings. These are defined next.

Definition 6. *Let p be the number of workers available in a system and let op be a concurrent operation. Assume that for each worker i, $i = 1, ..., p$, op_i is modeled as the affine mapping on x, $op_i(x) = x.op_i + OP_i$. Then, the average op mapping is defined to be*

$$op(x) = x.op + OP, \text{ where} \tag{3.13}$$

$$OP = \frac{\sum_{i=1}^{p} OP_i}{p} \text{ and } op = \frac{\sum_{i=1}^{p} op_i}{p}. \tag{3.14}$$

The average mappings of the operations $\{l_i : i = 1, ..., p\}$, $\{h_i : i = 1, ..., p\}$, $\{r_i : i = 1, ..., p\}$ and $\{w_i : i = 1, ..., p\}$ are denoted l, h, r and w, respectively.

3.2 Statement of the Scheduling Problem

In a StarAffine network, the make-span minimization problem is a difficult one. Although several theoretical results have been published [28, 36, 43, 44] no significant breakthroughs have been made an the problem remain NP-hard in most practical situations [13]. In [36] it is stated that the complexity of determining the optimal make-span for a general star platform is not known. This problem remain open until today. The complexity of the DLS problem is known in the case of linear communication and computation costs or in the case of homogeneous platforms [23]. In [41] the NP-completeness of a particular case of the DLS problem over heterogeneous platforms and with affine cost is proved.

Using the notation established in the previous sections, the scheduling problem can be formalized as following: given a StarAffine network $P = \{P_0, P_1, P_2, ..., P_p\}$, a set of core tasks X, and a the Mapping Group $MG = e, s_i, l_i, r_i, w_i$, decide how agglomerated the core tasks and when send these agglomerated task. Let $time(op, i, j)$ be the time spent from the beginning of execution of the job up to the time when operation op ends processing x_i agglomerated core tasks in jth round. Then the problem is minimizing the make-span, this is, solving

$$\underset{ij}{\text{Minimum}} \quad time(w_i, i, j) \tag{3.15}$$

$$\text{Subject to} \tag{3.16}$$

$$time(w_i, i, j) \geq \max\{time(s_i, i, j) + l_i(x_i), time(l_i, i, j), time(w_i, i, j)\} \tag{3.17}$$

$$+(r_i + w_i)(x_i) \tag{3.18}$$

$$x_i \geq 0 \tag{3.19}$$

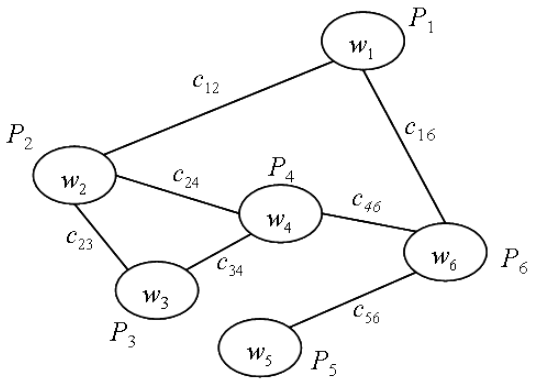

Figure 3–6: Platform Graph Model

the maximum in the before expression ensure that the begin to compute a tasks after receive completely the core tasks and the worker process only one tasks to a give time.

The alternative pursued in this dissertation is focused in the optimization of the throughput of each period of the schedule. As stated before, the problem of finding the best schedule can be solved in polynomial time if the mappings under underlying model are all linear [16]. The problem of finding the schedule that achieves the optimal throughput in each period can be formalized as follows:

Definition 7. *(THROUGHPUT DIVISIBLE). Let $K > O$, $\rho > 0$ and assume consider a system modeled by a* StarAffine *platform. Is there a periodic schedule that executes K load units every T time-units and such that $\frac{K}{T} > \rho$?.*

The above mentioned polynomial time solution for solution when the underlying mappings are all linear is discussed in section 3.3. if, on the other hand, the mappings underlying the model are affine, an asymptotically optimal algorithm for finding the best schedule, is introduced in [17]. In [21], a variant of the algorithm in [17] which allows for a computation of the optimal number of rounds, is presented. Section 3.4 discusses this approach.

3.3 Bandwidth-Centric Principle

3.3.1 Overview

In [16], a framework for solving the Master-Slave scheduling problem on a heterogeneous platform for a significantly large number of tasks, is introduced. The heterogeneous platform is modeled as a non oriented graph as the one shown in Figure 3–6. In this model, each processor is a node and each edge is a physical interconnection link. The tasks consist of identical and independent jobs. Each task is associated with a file that contains the data required for the task execution. Therefore, the model is slightly more general than the (StarAffine) network.

The master determines the number of core tasks and send operations necessary to distribute the tasks across the network according to the *bandwidth-centric* principle discussed below. Upon reception, nodes compute their segment of the received package, and forward the remaining tasks to the neighboring processors following the same *bandwidth-centric* policy.

The optimal steady state is determined by the fraction of time that the system spends computing and the fraction of time the system spends sending or receiving tasks along each communication link. This distribution is made in a way such the averaged of the total number of tasks processed at each time step is maximum.

3.3.2 Mathematical formulation

The above problem is modeled as a linear programming problem whose objective function maximizes the throughput. The parameters used for this formulation are the following:

- w_i, referred as the weight of the node P_i, represents the units of time that required to process one task.
- c_{ij}, referred as the weight of the edge e_{ij} between the nodes P_i and P_j, represents the time needed to communicate one task from processor P_i to processor P_j or vice versa.

The system is assumed to conform to the *full overlap, single-port* model. Next the theoretical framework developed in [16] is discussed in the context of a Star network.

3.3.3 Optimal steady-state problem formulation

A star network F admits a graph representation of the form depicted in Figure 1–1. The network consist of $p+1$ processors, where P_0 is the master and $\{P_1, P_2, ..., P_p\}$, are p workers. The master does not process tasks. The parameter c_{0j}, which represents the only communication between the master and processor P_i, is written simply as c_i.

In the context of a star network, the problem of finding the schedule with optimal throughput in each period is formulated as follows: Master Slave Scheduling Problem MSSG(G)

$$\text{Maximum} \qquad Ntask(F) = \sum_{i=1}^{p} \frac{\alpha_i}{w_i} \tag{3.20}$$

$$\text{Subject to} \qquad \sum_{i=1}^{p} s_i \leq 1 \tag{3.21}$$

$$\frac{\alpha_i}{w_i} = \frac{s_i}{c_i}, \text{ for } 1 \leq i \leq p \tag{3.22}$$

$$0 \leq \alpha_i \leq 1, \text{ for } 1 \leq i \leq p \tag{3.23}$$

$$0 \leq s_i \leq 1, \text{ for } 1 \leq i \leq p \tag{3.24}$$

where

- α_i, referred as the (average) fraction of time spent each time unit by P_i computing tasks.
- s_i, referred as the (average) fraction of time spent each time unit by P_0 sending tasks to processor P_i.
- r_i,referred as the fraction of time spent each time unit by P_i receiving tasks from the master P_0.

The number of tasks sending by P_0 to processor P_i is the same receive by P_i, $r_i = s_i$

The above linear programming problem is solved by using rational programming. The values returned by the linear program are α_i and s_i. These values are rational numbers. The period T of the schedule is the common denominator of the fractions $1/c_i$,α_i and s_i; $1 \leq i \leq p$. As a result, each processor P_i executes exactly $\frac{\alpha_i T}{w_i}$ tasks.

3.3.4 Formal results

Theorem 1. *With the above notations, the minimal value of ntask(F) for the star graph F is obtained as follows:*

1 Sort the worker by increasing communication times. Renumber them so that $c_1 \leq c_2 \leq ... \leq c_p$.

2 Let q be the largest index so that $\sum_{i=1}^{q} \frac{c_i}{w_i} \leq 1$. *if* $q < k$, *let* $\epsilon = 1 - \sum_{i=1}^{q} \frac{c_i}{w_i}$; *otherwise, let* $\epsilon = 0$

3 Then, $n_{task}(F) = \sum_{i=1}^{q} \frac{c_i}{w_i} + \frac{\epsilon}{c_{q+1}}$.

From Theorem 1 is concluded that the children cannot consume more tasks than those that the master processor can send. According to this result, a slow processor with a fast communication is better than a fast processor with a slow communication link.

3.4 Asymptotic performance, star network, affine costs

In [17], the Bandwidth-centric principle is applied to search for a solution to a DLT problem. An asymptotically optimal schedule for a model based on affine mappings is introduced. The next discussion combines results developed in [20] and in [21].

The proposed algorithm divides the overall processing time ν into n periods of length T. As in the previous section initial and final phases are ignored. The main problem is in this case, the selection of resources. The selection rule presented in [20] is similar to the one in Theorem 1.

3.5 Asymptotic Optimality

This section discusses the proof of the asymptotic optimality of the schedule introduced in [17]. Since the model is affine, it is necessary to find a optimal subset of workers. Let $I \in \{1, ..., p\}$ be a subset representing the participating workers. Then $\bar{x}$ must satisfy the following restrictions:

$$\sum_{i \in I} h_i(x_i) \leq T \tag{3.25}$$

$$\max \{(r_i + w_i)(x_i) | i \in I\} \leq T \tag{3.26}$$

where 3.25 restricts the communication and 3.26 restricts the computation time of each worker to a period T. The main aim is to maximize the average number of tasks processed in a period. Average task is defined as: $\textsc{AverageMaxTaks}(T) = \frac{\textsc{MaxTaks}(T)}{T} = \frac{\sum_{i \in I} x_i}{T}$. The solution is based in two problems. The first one is:

$$\text{Max} \quad \sum_{i=1}^{p} y_i \tag{3.27}$$

$$\text{Subject to} \quad \sum_{i=1}^{p} h_i.y_i \leq 1 - \tau \tag{3.28}$$

$$\max \{(r_i + w_i).y_i | 1 \leq i \leq p\} \leq 1 - \tau \tag{3.29}$$

where $\tau = \frac{\sum_{i=1}^{p}(H_i + R_i + W_i)}{T}$. The value τ restricts the problem described in equations (3.28) and (refeq2). As a result, the solution of this problem is such that $\sum_{i=1}^{p} y_i \leq \textsc{AverageMaxTaks}(T)$. Now, the solution to the problem described in equations (3.28) and (refeq2) follows directly from theorem 1,

$$\sum_{i=1}^{p} y_i = (1 - \tau)(\sum_{i=1}^{q} \frac{1}{r_i + w_i} + \frac{\epsilon}{c_{q+1}}) \tag{3.30}$$

Now, let $\textsc{MaxTaks}_{opt}(1)$ be the solution of the original problem given by thereon 1. Then,

$$\textsc{AverageMaxTaks}_{opt}(1) \leq \sum_{i=1}^{q} \frac{1}{r_i + w_i} + \frac{\epsilon}{h_{q+1}} \tag{3.31}$$

Let μ_{opt} the optimal time to process the X core tasks and μ the time necessary to process with the proposed algorithm; then

$$\mu_{opt} \geq \frac{X}{\sum_{i=1}^{q} \frac{1}{r_i+w_i} + \frac{\epsilon}{h_{q+1}}}. \tag{3.32}$$

If AverageMaxTaks(T) are processed during $n-1$ periods, then n satisfies MaxTaks(T)$(n-1) \geq X$. Let $n = \lceil \frac{X}{\text{MaxTaks}(T)} \rceil + 1$, $T = \sqrt{\mu_{opt}}$, and $2\tau \leq 1$. Then,

$$\mu < \mu_{opt} + 2\tau \frac{\mu_{opt}}{T} + 2T \tag{3.33}$$

Finally, the proof of the asymptotic optimality of the algorithm is completed by setting $T = \sqrt{\mu_{opt}}$, and obtaining

$$\mu \leq \mu_{opt} + 2(\tau + 1)\sqrt{\mu_{opt}} = \mu_{opt} + O(\sqrt{\mu_{opt}}) \tag{3.34}$$

3.5.1 Formal results

The next Theorem, which is written in terms of the terminology adopted in this thi dissertation, summarizes the previous discussion.

Theorem 2. *For arbitrary values of H_i, h_i, W_i and w_i the previous periodic multi-round algorithm is asymptotically optimal. Closed-form expressions for resource selection and task assignment are provided by the algorithm, whose complexity does not depend upon the total amount of work to execute.*

3.5.2 Periodic scheduler

In [21], the problem proposed in section 3.3.3, is reconsidered as:

$$\text{Max} \qquad \rho = \sum_{i=1}^{p} y_i \tag{3.35}$$

$$\text{Subject to} \qquad \sum_{i=1}^{p} h_i.y_i \leq 1 \tag{3.36}$$

$$\max\{(r_i + w_i).y_i | 1 \leq i \leq p\} \leq 1. \tag{3.37}$$

As remarked in the previous section, the solution obtained with Theorem 1 $\bar{y}_i$ is used as an upper bound. This upper bound is used to define $x_i = y_i \frac{X}{n}$ and computing the optimal value of n. Then, the period becomes

$$T = \max\{\max\{(r_i + w_i)(y_i) | 1 \leq i \leq q\}, \sum_{i=1}^{q} h_i(y_i) = \max(\frac{a}{n} + b, \frac{a'}{n} + b') \quad (3.38)$$

where $a = X.\sum_{i=1}^{q} \frac{y_i h_i}{\rho}$, $b = \sum_{i=1}^{q} H_i$, $a' = X.\max\{(r_i + w_i).y_i | 1 \leq i \leq q\}$ and $b' = 0$.

Two cases are considered when finding a maximum. These are:

If $n > \frac{a'-a}{b}$ then $\frac{a}{n} + b > \frac{a'}{n}$ and the make-span is:

$$\mu = (n+1)T = a + b + \frac{a}{n} + bn, \text{ which is minimized for } n = \sqrt{\frac{a}{b}} \quad (3.39)$$

If $n \geq \frac{a'-a}{b}$ then $\frac{a}{n} + b \geq \frac{a'}{n}$ and the make-span is:

$$\mu = (n+1)T = a' + \frac{a'}{n}, \text{ which is minimized for } n \text{ large as possible} \quad (3.40)$$

The optimal number of rounds for such a periodic schedule is therefore,

$$n = \max\{\sqrt{\frac{a}{b}}, \frac{a'-a}{b}\} \quad (3.41)$$

3.6 UMR scheduler

UMR is a multi-round, non-periodic algorithm for scheduling divisible tasks on parallel computing systems. According to [21] "The idea behind UMR is simple: assign chunks of "uniform" sizes to all workers within each round, increasing the chunk size between rounds geometrically. Here "uniform" means that it takes the same amount of time for each worker to compute its chunk at each round". For homogeneous systems over a full overlap single port model and with a star topology, the method is designed to send to each worker the same amount of work in a round. There are two versions for the variation of this uniform amount. In the original version of the scheduler [45], called UMR, the uniform amount of work is

increased geometrically with each round. In a revised version, called UMR2[19, 46], the uniform amount is increased or decreased depending on a parameter θ, which depends on the number of workers chosen. On the other hand, the amount of work is increased if $\theta > 1$ and decreased if $\theta < 1$. UMR maintains perfect worker utilization throughout the execution. UMR2 instead, may not have a perfect worker utilization for $\theta < 1$. As remarked earlier, at the base of UMR and UMR2 is a set of affine equations of the form of (5.74) for expressing execution times in terms of load. These equations are similar in spirit to the one used for SCOW. UMR equations model solely communications and computations. These equations are:

$$Tcomp_i = cLat_i + \frac{chunk_i}{S_i}, \tag{3.42}$$

$$Tcomm_i = nLat_i + \frac{chunk_i}{B_i}; \tag{3.43}$$

where S_i is the amount of work completed in a unit of time, B_i is the bandwidth of the communication link, $cLat_i$ is the start time of the operation of computation, and $nLat_i$ is the start time of the operation of communication. The equations for UMR are guided by a simple perfect bandwidth utilization principle: "The time the last worker, labeled as the Nth worker, spends in receiving the last bytes of data, initiating a computation, and computing a data chunk during round j; must be equal to the time it takes for the master to send data to all the N workers during round j+1". This principle translate into an equation, which for a homogeneous cluster, is

$$cLat + \frac{chunk_j}{S} = N(nLat + \frac{chunk_j + 1}{B}) \tag{3.44}$$

This equation can be transformed in turn, into a simple induction over $chunk_j$, which yields a geometric or arithmetical series of chunk sizes, where $chunk_0$ is the only unknown. There is also a condition for full platform utilization which restricts the number of workers used. As the author remark "when the full platform cannot be utilized effectively: in the homogeneous case, one can just reduce the value of N,

for the heterogeneous case implements the resource selection inspired by the work in [27]".

The make-span minimization problem for UMR is stated as:

Minimize

$$Ex(M, chunk_0) = \frac{W_{total}}{N} + M \times cLat + \frac{N}{2} \times (nLat + \frac{chunk_0}{B}) \tag{3.45}$$

Subject to

$$G(M, chunk_0) = \sum_{j=0}^{M-1} N \times chunk_j - W_{total} = 0 \tag{3.46}$$

In a revised version, called UMR2, the uniform amount is increased or decreased depending on a parameter θ. If $\theta > 1$ the amount is increased; and decreased if $\theta < 1$. The UMR scheduler maintains perfect worker utilization throughout the execution, but if $\theta < 1$ there is no perfect worker utilization for URM2.

The algorithm for computing the parameters of UMR or UMR2 receives as inputs the constant values of these mappings and return the amount of work for the first round. Then, a recursive formula is applied to compute the increments or decrements per round. The amount of work returned is a real value. No *post-mortem* discretization is proposed in the literature. Instead, UMR and UMR2 methods concentrate on computing the optimal value for `nRounds`. UMR2 make no modifications in the last round for the workers to end operating at the same time.

CHAPTER 4
OPTIMIZATION CRITERIA FOR A PERIOD UNDER AFFINE MODEL

This chapter presents a formulation of the general scheduling problem under two different constrains. These are Maximal Production (MP) and Maximal Utilization (MU) in a period. An optimal solution and an approximate solution in a particular case of the problem are presented, as well.

In section 3.1, the three mayor segment times that can be distinguished within a period T are described. In fact, the period T is an upper bound for these segments. This upper bound restriction, in turn, produces a set of partitions which induce a division of the job into agglomerated core tasks. Such division depends directly on the time T allocated for a period. The next definition attempts to capture all these basic characteristics in a single concept.

Definition 8. *Let T and X be nonnegative real numbers, p a positive integer and $F = \{f_j : \ j = 1, ..., m\}$ a finite set of real-valued mappings defined in the real numbers. A (T, p, MG)-partition of X is a k-tuple $\bar{x} = (x_1, ..., x_p)$ that satisfies:*

$$\text{(a)} \quad \sum_{i=1}^{p} x_i \leq X \text{ and} \tag{4.1}$$

$$\text{(b)} \quad \text{For each } j = 1, ..., m \text{ and each } i = 1, ..., p; \ f_j(x_i) \leq T. \tag{4.2}$$

$$\text{(c)} \quad \text{For each } i = 1, ..., p; \ x_i \geq 0 \tag{4.3}$$

The word partition does not means that the whole data load X is split into mutually disjoin subsets since in a period the load processed is only part of the total load.

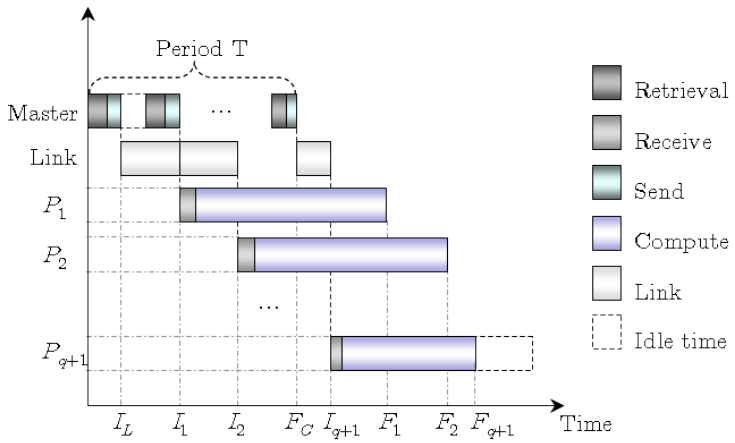

Figure 4–1: Pattern of Solution to MP, over Star Network.

4.1 Maximizing the production

The maximal production problem over a single period T considers a load size X large enough so all workers receive core tasks within this period. The problem is formulated in the next section.

4.1.1 Maximal production problem

Let x_i be the number of units of data load sent to worker P_i. The Maximal Production problem (MP) can be visualized as follows: Figure 4–1 is a Gantt Chart of the execution times of process under maximal production. In this chart I_i denotes the idle time of P_i and I_L denotes the idle time to star the communication over the link. The goal is to maximize the total number of processed load units, MaxTask(T)=$\sum_{i=1}^{p} x_i$, according to the model defined in Section 3.1. In Figure 4–1, only the $q \leq p$ selected workers participate in the computation. The envelop and send time is restricted to a period T (i.e. $\sum_{i=1}^{q}(e+s)(x_i) = F_C \leq T$). Data transmission time is also restricted to period T (i.e. $\sum_{i=1}^{q}(l_i)(x_i) = I_q - I_l \leq T$) as well as the receive and compute times (i.e. $(r_i + w_i)(x_i) = F_i - I_i \leq T$ for all i). The above problem description is independent of the load size X. This value X is used in chapter 5 to find a minimal make-span.

The following mixed linear program is used for computing the optimal resource selection and load distribution. The objective function is the number of tasks processed in the period T, χ_i is a boolean variable that equals 1 if the worker i participates in the problem solution.

$$\text{Maximize} \qquad \textsc{MaxTask}(\mathrm{T}) = \sum_{i=1}^{p} x_i \tag{4.4}$$

$$\text{Subject to} \qquad R = \sum_{i=1}^{p} \chi_i \ (e+s)(x_i) \leq T \tag{4.5}$$

$$L = \sum_{i=1}^{p} \chi_i \ l(x_i) \leq T \tag{4.6}$$

$$C_i = (r_i + w_i)(x_i) \leq T, \quad 1 \leq i \leq p \tag{4.7}$$

$$\chi_i \in \{0, 1\}, \quad 1 \leq i \leq p \tag{4.8}$$

$$(r_i + w_i).x_i \leq \chi_i T, \quad 1 \leq i \leq p \tag{4.9}$$

$$x_i \geq 0, \quad 1 \leq i \leq p \tag{4.10}$$

Equation (4.5) states that the time spent by the master in the envelop and send operations do not exceed the time period. Equation (4.6) states that the time spent by communicating data over the link do not exceed the time period. Equation (4.7) restricts receive and compute operations to the time period, and equation (4.9) states that no task is given to processors that do not participate in the solution (those for which $\chi_i = 0$). This mixed linear program is the most general formulation for the maximal production problem.

Theorem 3. *The optimal solution for the MP problem is given by the solution of the mixed linear program above.*

Proof: Direct consequence of problem definition.

The auxiliary mapping h defined in section 3.1.2 is introduced to reformulate the MP problem in such a way that the variable χ_i is eliminated and as well as one the original restrictions.

"Given a time T, find $I \subset \{1, 2, ..., p\}$ such that

$$\text{Max } \{\sum_{i \in I} x_i \quad : \quad \bar{x} \text{ is } (T, p, MG) - \text{partition of } X\} \tag{4.11}$$

$$\text{Subject to} \qquad \sum_{i \in I} h_i(x_i) \leq T \tag{4.12}$$

$$\max \{(r_i + w_i)(x_i) | i \in I\} \leq T\text{"}. \tag{4.13}$$

This formulation shows clearly the combinatorial nature of the solution since there is an exponential number of subset in {1,2,...,p}. However, when the subset is known, the solution is characterized in proposition 1.

In the particular case when the model is linear, the solution can be found in polynomial time. Assuming that $h_i(x_i) = s_i T$ and $(r_i + w_i)(x_i) = \alpha_i T$, the problem is transformed to the problem in section 3.3.3 and therefore, the solution follows directly from Theorem 1. If on the other hand, mapping $e + s$ dominates over mapping l_i, or if the platform is a *busnetwork*, the solution is obtained by sorting by compute capacities, as demonstrated in appendix A.1.

4.1.2 Optimal production over an homogeneous cluster

First, the solution for *homogeneous cluster* is described and analyzed. *Homogeneous cluster* as describe in section 3.1 consists of p identical workers accessible via a network link. Therefore, $l_i = l$; $r_i = r$; $w_i = w$ for each mappings $1 \leq i \leq p$, where l, r and w are average mapping in the sense of definition 6. The next Theorem provides a particular solution for the MP problem in this homogeneous platform.

Theorem 4. *Let T be a real nonnegative number and p be a positive integer. Let*

$$Y = (r + w)^{-1}(T); \tag{4.14}$$

$$q = \min \{p, \left\lfloor \frac{T}{h(Y)} \right\rfloor\} \tag{4.15}$$

and define

$$T_\epsilon \;=\; \begin{cases} T - q \times h(Y) \text{ if } q < p \\ 0, \text{ otherwise} \end{cases} \tag{4.16}$$

Then the maximal values to MaxTask(T) *is,*

$$\text{MaxTask(T)} = qY + \max\{0, h^{-1}(T_\epsilon)\} \tag{4.17}$$

Proof The proof is reduced to demonstrate the claims: (a) The tuple

$(Y, ..., Y, Y_{q+1}, 0, ..., 0)$ is a solution, and (b) $qY + Y_{q+1}$ is the maximum.

Claim (a) follows directly from basic properties and definitions. Indeed, $(r+w)(Y) = T$ this showing that Y satisfies (A.3) of MP for $i \leq q$. If $q < p$, then $h(Y_{q+1}) = T_\epsilon = T - q \times h(Y)$. From (4.15) it follows that $h(Y_{q+1}) \leq h(Y)$. Since h is a nondecreasing mapping, $Y_{q+1} \leq Y$. But then, since $r + w$ are also nondecreasing mappings, $(r + w)(Y_{q+1}) \leq (r + w)(Y) = T$. Therefore, Y_{q+1} satisfies (A.3) of MP, as well. To prove A.2, by definition (4.15), $\sum_{i=1}^{j} h(X_i) \leq T$ for all $j \leq q$. If $q < p$, $\sum_{i=1}^{q+1} h(x_i) = q \times h(Y) + h(Y_{q+1}) = q \times h(Y) + T_\epsilon = T$, by the definition of $T\epsilon$. This shows that the tuple is a solution of MP.

Claim (b) In order to show that this solution $\bar{Y}$ is maximal, let's consider the set of all optimal MP solutions. Let $\bar{X}'$ a tuple in this set and q' the number of worker in this solution.

if $q \geq p$, suppose $q' < p$ them $\{X'_1, ..., X'_{q'}, Y_p\}$ is a solution than process more task that the original one, a contradiction, then $q' = p$.

if $q < p$, suppose $q' > q + 1$ them $T = q \times h(Y) + h(Y_{q+1}) = h(q \times Y + Y_{q+1}) + (q+1)H < h. \sum_{i=1}^{q'} X'_i + q'H$, contradiction because $\bar{X}'$ must satisfies A.2. Suppose $q' < q+1$ then $qY + Y_{q+1} \leq \sum_{i=1}^{q'} X'_i$, contradiction because no is possible distribute $qY + Y_{q+1}$ core tasks in q workers, then boot solution use the same number of processor $q' = q + 1$.

Suppose, without loss of generality, that the two solution use the same subgroup of processor, them $T = h(q \times Y + Y_{q+1}) + (q+1)H \leq h.\sum_{i=1}^{q+1} X'_i + (q+1)H$, finally derive that $q \times Y + Y_{q+1} = \sum_{i=1}^{q+1} X'_i$, this conclude that $qY + Y_{q+1}$ is the maximum. □

The solution of MP obtained in this Theorem reveals that there exits the possibility of a continuous processing in some workers. A particular solution is stated as follows: q workers execute Y agglomerated core tasks continuously. If in addition $\epsilon = 0$, all concurrent tasks are executed by these q workers making the continuity strong. If $\epsilon > H$, an additional worker is required to execute $Y_{q+1} = \max\{0, h^{-1}(T_\epsilon)\}$ remaining core tasks that the master is still able to submit within the given T units of time. In this case, the continuity between periods is weak, although all but one of the workers operate continuously. the next corollary present this particular solution.

Corollary 1. *The following (T,p,MG)-partition of X is a optimal solution of MP problem is,*

$$x_i = Y = (r+w)^{-1}(T) \text{ for each } i = 1, ..., q; \tag{4.18}$$

$$x_{q+1} = \max\{0, h^{-1}(T_\epsilon)\}, \text{and} \tag{4.19}$$

$$x_i = 0 \text{ for } i > q+1. \tag{4.20}$$

Due to the characterization given in Theorem 4, if $\epsilon > H$, the solutions are the infinite set $\{\bar{x} \mid \sum_{i \in I} x_i = qY + \max\{0, h^{-1}(T_\epsilon)\} \text{ and } |I| = q+1\}$. The next Theorem guarantees an upper bound to the number of required workers. This number is independent of T.

Theorem 5. *Let $T \geq 0$ be a real variable and i a variable ranging over the set of all positive integers. Let*

$$R(i, T) = \sum_{i=1}^{i} h(Y) \tag{4.21}$$

be the time spent by the master process in completing a round of data installments for i workers so condition (4.14) is achieved. Let $\lambda = \frac{r+w}{h}$ be such that $\lambda H > R + W$.

Then, there is a positive integer $\bar{p} = \lfloor \lambda \rfloor$ *such that the equation*

$$R(i,T) = T, \tag{4.22}$$

(a) has a unique solution (i, T_i) *for each* i, $1 \leq i \leq \bar{p}$, *and*

(b) $R(i,T) > T$ *for all* $i > \bar{p}$.

Proof. Let

$$R(i,T) = \sum_{j=1}^{i} h(Y) \tag{4.23}$$

Let $\lambda = \frac{r+w}{h}$ be such that $\lambda H > R + W$ and let $\bar{p} = \lfloor \lambda \rfloor$.

From (4.14), $Y = (r+w)^{-1}(T) = \frac{T-(R+W)}{r+w}$. Consequently,

$$R(i,T) = \sum_{i=1}^{i} h(Y) = a(i)T + b(i) \text{ where} \tag{4.24}$$

$$a(i) = \frac{i}{\lambda} \text{ and} \tag{4.25}$$

$$b(i) = i(h - \frac{R+W}{\lambda}) \tag{4.26}$$

Proof of (a): Since $\lambda H > R + W$ the intersection between mapping (4.24) and the y-axis is positive. The slope of mapping (4.24) is $0 < a(i) \leq 1$ if $i \leq \lambda$. Therefore, $R(i,T) = T$ has a solution $T_i = \frac{i(\lambda H - (R+W))}{\lambda - i}$, and this solution is nonnegative for $i \leq \bar{p}$.

Proof of (b): For $i > \bar{p}$ the values of the slope and the y-axis intercept of mapping (4.24) are $a(i) > 1$ and $b(i) > 0$, respectively. Therefore, $R(i,T) = T$ has only negative solution. Consequently, $R(i,T) > T$ for all $T \geq 0$.□

It is worth remarking that if $\lambda H < R + W$, the time cost of the overhead in the computation does not allow computations within a period $T \leq T_\lambda = \lambda H$. In this case $T_i < 0$ if $i \leq \bar{p}$. Also, if $i > \bar{p}$, $R(i,T) = T$ has a positive solution. But this solution does not guarantee a positive value for $Y = \lambda \frac{iH-(R+W)}{(\lambda-i)(r+w)}$ except for $iH < R+W$. In this case, perfect utilization only occurs within the index $\bar{p} < i \leq j$ where $j = \lfloor \frac{R+W}{H} \rfloor$.

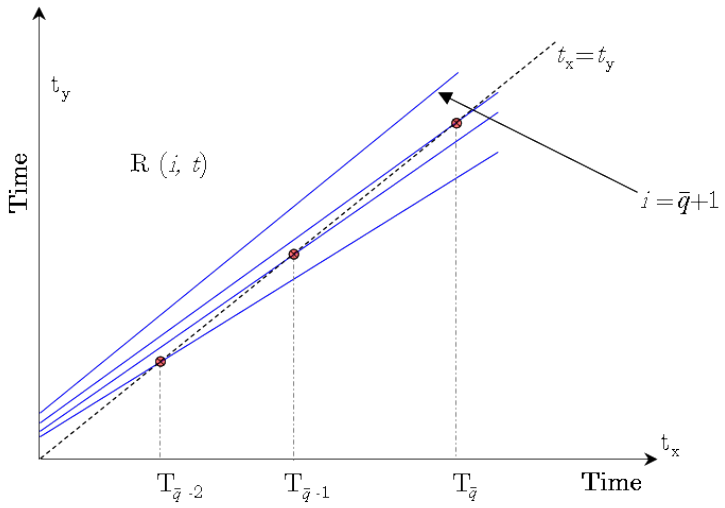

Figure 4–2: $R(i,T)$ Graph Representation.

Figure 4–2 pictures different plots of the functions $R(i,t)$. The plots range up to $i = \bar{p}+1$. In this figure the values T_i represent, right limit for the time T requiring exactly i workers in the solution of MP. If the slope of $R(i,t)$ is greater than or equal to 1 the equation 4.22 has no solution. In summary, this Theorem states that there are balanced schedules in the solution of MP provided by Theorem 4. Each such balance schedule occurs whenever T is chosen such that $\epsilon < H$.

4.1.3 Optimal production over an heterogeneous cluster

All results in this section refer to the case when mapping $h_i = l_i$. Figure 4–2 shows the dependence between resource selection and the size of time period T. Furthermore, as presented in section 4.1.1, the difficulty arises in solving the resource selection problem, specially when the period is small. When latencies are introduced, it is difficult to decide which resources to use. In the next example the optimal solution for three different values of T are presented.

Example 1. In the toy example of Figure 4–3 the optimal solution for three periods $T = 7, 9$ and 27, are computed by using the standard solver Lindo [47]. With period $T = 7$, Figure 4–3(b) shows link comparisons between the best possible subset solutions {1,2} and {3}. The optimal solution uses the last processor P_3 and the maximal production is 3 with values $\bar{x} = \{0, 0, 3\}$. However, if the period is $T = 9$

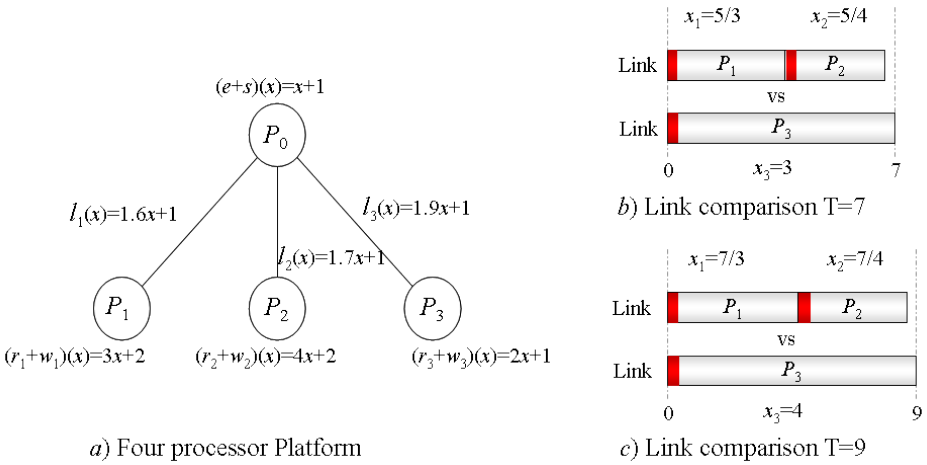

Figure 4–3: Example with Four Processor.

Figure 4–3(c) shows that the solution with maximal production is 4.083 with values $\bar{x} = \{2.333, 1.75, 0\}$. Furthermore, with a period $T = 27$, the solution uses the whole platform. The maximal production is 14.605 with values $\bar{x} = \{8.333, 6.25, 0.219\}$.

These solutions with different values of T show a direct dependence between the subset of workers and T. In fact, Figures 4–3(b and c) show the difficulty in resource selection. Such difficulty involves solving a trade off between selecting the resource with faster link capacity or the resource with more effective task transmission per time unit. The latter concept is illustrated by the fact that link 3 can send 3 tasks in 7 unit time while link 1 and link 2 together, can only send 2.916 tasks, due to the compute restriction. Nonetheless, link 1 and link 2 are faster than link 3. In contrast, when using $T = 9$, the link 1 and link 2 together are better that link 3. This holds in all periods greater than $T = 8$. After $T = 27$, the resource selection remains constant. This selection agrees with selection provided by Theorem 1.

If the time period is large enough, it is possible to derive an approximate solution which can be used to design an asymptotically optimal scheduler. In addition a bound to this approximation, which is independent of the time period, is derived in this section.

Proposition 1. *In any optimal solution of the MP problem, all participant workers are fully use, except perhaps for the worker with maximal link capacities* g_i.

Proof. Let i, the index for the worker with maximal link capacity. Suppose an optimal solution in which there exists a worker $j \neq i$ not fully used. This means $(r_j + w_j)(x_j) < T$. It is possible to derive a solution that performs at least as many tasks as those in the optimal solution. Let τ be a smaller fraction of time, $\beta_j = \frac{\tau}{h_j}$ and $\beta_i = \frac{\tau}{h_i}$. The communication time does not change if the vales x_j and x_i are replace with $x_j + \beta_j$ and $x_i - \beta_j$, respectively. Hence, there is an optimal solution which processes $\beta = \tau(\frac{h_i - h_j}{h_i h_j})$ more tasks than the original one. This is a contradiction. Therefore, all workers with link capacity smaller than the workers with maximal link capacity are fully used. □

The above proposition ensures that in an optimal solution all processors work to full capacity except perhaps by the processor with lower speed communication.

The next Theorem is a modification of a similar result presented in [20] for a single round StarAffine divisible load problem. This Theorem is the main result in this section.

Theorem 6. *If the period is large enough, then for the maximal production is obtain saturates the processor in the order of non decreasing link capacities h_i.*

Proof. Consider a valid solution of the MP problem for a time period T and MaxTask(T) is the optimal number core task that can be processed with this solution.

- Consider the following instance of the MP problem, with p workers where for all i, $H_i^{'} = 0$, $R_i^{'} = 0$, $W_i^{'} = 0$, $h_i^{'} = h_i$, $r_i^{'} = r_i$, $w_i^{'} = w_i$ and $T^{'} = T$. Since all computation and communication latencies have been eliminated, the optimal number of core task $\text{MaxTask}_1(T)$ processed by this instance is larger than the number of core task $\text{MaxTask}(T)$ processed by the initial platform. From Theorem 1, the value of $\text{MaxTask}_1(T)$ is given by a formula

$$\text{MaxTask}_1(T) = f(I, \sigma)T. \tag{4.27}$$

where $f(I,\sigma) = \sum_{i=1}^{q} \frac{h_i}{w_i} + \frac{\epsilon}{h_{q+1}}$. The set I of $q+1$ workers, the send order σ and the value of ϵ are given by Theorem 1.

- Consider now an instance of the MP problem with p workers where for all i, $H_i^{'} = 0$, $R_i^{'} = 0$, $W_i^{'} = 0$, $h_i^{'} = h_i$, $r_i^{'} = r_i$, $w_i^{'} = w_i$ and $T^{'} = T - \sum_{i \in I}(H_i + R_i + W_i)$. Clearly, the optimal number of task $\textsc{MaxTask}_2(T)$ processed in this instance is lower than $\textsc{MaxTask}(T)$, since it adds all the communication and computation latencies before the beginning of the processing. Moreover, as before $\textsc{MaxTask}_2(T)$ is given by the formula

$$\textsc{MaxTask}_2(T) = f(I,\sigma)(T - \sum_{i \in I}(H_i + R_i + W_i)). \tag{4.28}$$

Therefore,

$$f(I,\sigma)\left(1 - \frac{\sum_{i \in I}(H_i + R_i + W_i)}{T}\right) \leq \frac{\textsc{MaxTask}(T)}{T} \leq f(I,\sigma). \tag{4.29}$$

Hence, when T becomes arbitrarily large, then the throughput, $\frac{\textsc{MaxTask}(T)}{T}$, becomes arbitrarily close to $f(I,\sigma)$, which is the optimal throughput if there were no communication and computation latencies. □

Intuitively, the effect of latencies in the solution can be bypass when the period is large enough. The main difference in the selection sets of resources arises in the time spent in communication latencies. The next extension of Theorem 1, summarizes this effect.

Theorem 7. *Let p be a positive integer and* $\textsc{MaxTask}(\mathrm{T})$ *be the optimal solution of the previous problem. The method:*

1. *Sort the worker by increasing communication times. Renumber them so that $h_1 \leq h_2 \leq ... \leq h_p$.*
2. *Let $Y_i = (r_i + w_i)^{-1}(T)$ for $1 \leq i \leq p$ and q be the largest index so that $\sum_{i=1}^{q} h_i(Y_i) \leq T$. if $q < p$, let $T_\epsilon = T - \sum_{i=1}^{q} h_i(Y_i)$; otherwise, let $T_\epsilon = 0$*

Returns the values to construct the following inequalities:

i)

$$\left|\textsc{MaxTask}(\mathrm{T}) - \sum_{i=1}^{q} Y_i + \max\{0, h_{q+1}^{-1}(T_\epsilon)\}\right| \leq \frac{\sum_{i=1}^{q} H_i}{h_{q+1}} \tag{4.30}$$

ii) If the period T is large enough

$$\left|\textsc{MaxTask}(\mathrm{T}) - \sum_{i=1}^{q} Y_i + \max\{0, h_{q+1}^{-1}(T_\epsilon)\}\right| \leq \frac{H_{q+1}}{h_{q+1}} \tag{4.31}$$

Proof. The prof is similar to the one in Theorem 6. The first instance of the MP problem is changed by the following: for all i, $H_i^{'} = 0$, $R_i^{'} = R_i$, $W_i^{'} = W_i$, $h_i^{'} = h_i$, $r_i^{'} = r_i$, $w_i^{'} = w_i$ and $T^{'} = T$. The second instance of the MP problem is the original. Both instances are solved with the previous method. Then, the inequalities are obtained with a equation that is similar to 4.29.

The previous Theorem solves partially the MP problem for a large period. The solution is partial because a piece of it has again a new resource selection problem similar to the first one, but with a smaller period T_ϵ. This period T_ϵ can be small enough to see the effects of latency dependences.

4.2 Maximazing the utilization

In this section a new metric for system utilization is defined. This metric is built to be maximal when the system is perfectly balanced. As in section 4.1, the Maximal Utilization (MU) problem is formulated over a single period T with a load size X large enough so all workers receive core tasks within this period.

4.2.1 Maximal utilization problem

The concept of system utilization is introduced here.

Definition 9. *Given a (T, p, MG)-partition $\bar{x} = (x_1, x_2, ..., x_p)$ of a number X of agglomerated tasks, where $F = MG$. The utilization differentials are defined as:*

$$\Delta_h(T, p, (x_1, ..., x_p)) = T - \sum_{\{x_i \neq 0\}} h_i(x_i) \text{ for the master or network} \tag{4.32}$$

$$\Delta_c(T, p, (x_1, ..., x_p)) = \sum_{\{x_i \neq 0\}} (T - (r_i + w_i)(x_i)) \text{ for the workers.} \tag{4.33}$$

The partition is assumed to the rearranged in such a way that there is an index k such that $x_i \neq 0$ for $i = 1, ..., k$ and $x_i = 0$ for $k < i \leq p$. Then, the master or network utilization in a period T is

$$U_h = \frac{\sum_{i=1}^{k} h(x_i)}{T} = 1 - \frac{\Delta_h(T, p, (x_1, ..., x_p))}{T}. \tag{4.34}$$

The utilization of the workers, in turn, is defined as

$$U_c = \frac{\sum_{i=1}^{k} (r_i + w_i)(x_i)}{kT} = 1 - \frac{\Delta_c(T, p, (x_1, ..., x_p))}{kT}. \tag{4.35}$$

The utilization in a round is thus,

$$U = U_h + U_c = 2 - \frac{1}{T}\left(\Delta_h(T, p, (x_1, ..., x_p) + \frac{1}{k}\Delta_c(T, p, (x_1, ..., x_p))\right). \tag{4.36}$$

The MU problem is stated as follows:

"Given a time T, find $I \subset \{1, 2, ..., p\}$ such that

$$\text{Max } \{U \quad : \quad (T, p, MG) - \text{partition of } X\} \tag{4.37}$$

$$\text{Subject to } H \quad = \quad \sum_{i \in I} h_i(X_i) \leq T \tag{4.38}$$

$$C \quad = \quad \max \{(r_i + w_i)(X_i) | i \in I\} \leq T." \tag{4.39}$$

Clearly, MU problem is equivalent to find the same subset to minimizing

$$\Delta = \Delta_h(T, p, (x_1, ..., x_p) + \frac{1}{k}\Delta_c(T, p, (x_1, ..., x_p)) \tag{4.40}$$

subject to the same restrictions. The solution of these problems depends, in turn, on the MP solution, section 4.1, to homogeneous system. However, this is not true for heterogeneous systems. In heterogeneous systems this problem is independent of the maximal production problem since it is possible find a maximal utilization that equals 2, but with a poor production.

4.2.2 Optimal utilization over an homogeneous cluster

The solution of the MP problem found in Theorem 4 is used to solve MU as shown in the next Theorem.

Theorem 8. *Let* $(x_1, ..., x_p)$ *be the solution to MP given by Lema 1. Then,*

$$\Delta = \begin{cases} \min\ \{\Delta_h(T, q, (x_1, ..., x_p)), \frac{1}{q+1}\Delta_c(T, q+1, (x_1, ..., x_{q+1}))\} \text{ if } q < p \\ \Delta_h(T, p, (x_1, ..., x_p)), \text{ otherwise} \end{cases} \tag{4.41}$$

solves MU.

Proof Let $(x_1, ..., x_k)$ be the solution of MP given by Theorem 4. Assuming that $q < p$; the proof is reduced to demonstrating the claims:

(a) The tuple $(x_1, ..., x_k)$ is a solution to MU and

(b) $\min\ \{\Delta_h(T, q, (x_1, ..., x_q)), \frac{1}{q+1}\Delta_c(T, q+1, (x_1, ..., x_{q+1}))\}$ is the minimum.

Claim (a) follows directly from the fact that both problems, MP and MU, have the same restrictions.

Claim (b). Let $\bar{Z}$ be a tuple that solves MU and without loss generality, let $1, ..., k_z$ the workers use in the solution.

if $\sum_{i=1}^{k_z} Z_i < \sum_{i=1}^{q} x_i$, then

$$\begin{aligned} &\Delta_h(T, k_z, (Z_1, ..., Z_k)) > \Delta_h(T, q, (x_1, ..., x_k)) \\ &\geq \min\ \{\Delta_h(T, q, (x_1, ..., x_q)), \frac{1}{q+1}\Delta_c(T, q+1, (x_1, ..., x_{q+1})). \end{aligned} \tag{4.42}$$

This is a contradiction, as well. Thus, the only possibility left is

$$\sum_{i=1}^{q} x_i \leq \sum_{i=1}^{k_z} Z_i \leq \sum_{i=1}^{q+1} x_i \text{ and } k_z \geq q \tag{4.43}$$

Suppose that $\sum_{i=1}^{q} x_i < \sum_{i=1}^{k_z} Z_i < \sum_{i=1}^{q+1} x_i$, them $k_z > q$, after algebraic manipulations is concluded that $\frac{1}{q+1}\Delta_c(T, q+1, (x_1, ..., x_{q+1})) < \frac{1}{k_z}\Delta_c(T, k_z, (Z_1, ..., Z_{k_z}))$, therefore $\frac{1}{k_z}\Delta_c(T, k_z, (Z_1, ..., Z_{k_z}))+\Delta_h(T, k_z, (Z_1, ..., Z_{k_z})) > \frac{1}{q+1}\Delta_c(T, q+1, (x_1, ..., x_{q+1}))$, which is again a contradiction. Therefore, $\sum_{i=1}^{k} Z_i = \sum_{i=1}^{p} X_i$ or $\sum_{i=1}^{k} Z_i = \sum_{i=1}^{q+1} X_i$.

this induce the following selection

$$\Delta = \min \{\Delta_h(T, q, (x_1, ..., x_p)), \frac{1}{q+1}\Delta_c(T, q+1, (x_1, ..., x_{q+1}))\} \tag{4.44}$$

because in boot extreme ones of Δ_h or Δ_c is null.

Assuming now that $q \geq p$ and $\bar{Z}$ is again the tuple that solves MU.

If $\Delta_h(T, k_z, (Z_1, ..., Z_{k_z})) > \Delta_h(T, q, (x_1, ..., x_q)$ then, $\Delta_c(T, k_z, (Z_1, ..., Z_{k_z})) > 0$, let $\rho > 0$ small enough, now any Z_i satisfying $(r + w)(Z_i) < T$ can be replaced with $Z_i + \rho$ and get a better solution. This contradicts the maximality of $\bar{Z}$. Therefore, $\Delta_h(T, k_z, (Z_1, ..., Z_k)) = \Delta_h(T, q, (x_1, ..., x_q)$. $\square$

If Theorem 4 gives $q < p$ workers and $\Delta_c = 0$; then perfect utilization and maximal production are both achieved. On the other hand, if Theorem 4 yields $p + 1$ workers, then $\Delta_c \neq 0$ and therefore, only maximal production is achieved. If $q < p$ then $\Delta_c = 0$. In this case the utilization is perfect but production is not maximal. In general, under maximal production, the worker's utilization is bounded by $1 - \frac{1}{k} < U_c \leq 1$. Similarly, under perfect worker's utilization, the master or communication network's utilization is bounded by $1 - \frac{1}{k} \leq U_h \leq 1$.

Intuitively, this Theorem states that there is a trade off between completely saturating the link or the master; or eliminating the gaps in the workers executions. Unfortunately, saturating master, links and workers is achievable only in a finite set of values, just as described for homogeneous systems in Theorem 5. Furthermore, in a system with limited resources, the solution of the maximal utilization problem is easy, since the maximum is achieved when there are not idle time in the workers' computations.

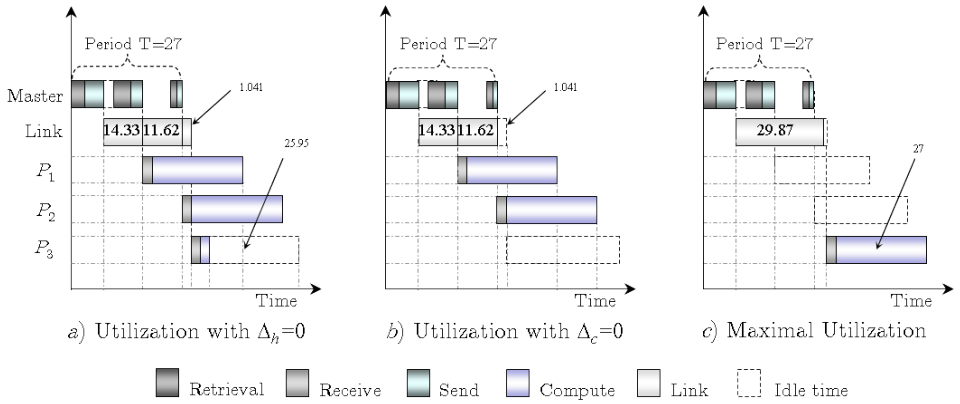

Figure 4–4: Example of Utilization with Four Processor.

4.2.3 Approximate optimal utilization over an heterogeneous cluster

The Figure 4–4, shows an example of independence between maximal production and maximal utilization problems over an heterogeneous platform. This independence is characterized by the possibility of saturating the link and some resources so that the utilization is maximal but the production is very poor.

Example 2. In this example it is supposed that the platform of the Figure 4–3(a), but with no computation overhead in processor P_3. The optimal solution is computed for period $T = 28$ using the standard solver Lindo [47]. The maximal is given by the workers selection done in Figure 4–4(c). In this solution $\Delta_h = 0.04$ and $\Delta_c = 0$. Then, the maximal utilization is $U = 1.98$ but this solution has a production of MaxTask(28)=14. An alternative problem is to find the maximal utilization with a fixed subset of workers with a previously set activation order. Figure 4–4(a and b) shows two possibilities for getting maximal utilization restricted to a previous solution that already had maximal production. This solution is obtained by using the same policy of Theorem 8 for the homogeneous networks. The solution was obtained by taking the minimum $\Delta = \min\{1.08, 9.3\}$. Then, the utilization is $U = 1.96$ and MaxTask(28)=15.16

If the solution of the MP problem is given, the Theorem 8 can be used to induce an approximate solution to the MU problem in heterogeneous system.

4.3 Maximal production and perfect utilization

The Perfect Utilization and Maximal Production problem (PUMP) is the problem of finding the pairs (i, T) where $0 < i$ is a number of workers and T is a period, in which the utilization is perfect and the production is maximal with i workers.

4.3.1 Optimal utilization-production over an homogeneous cluster

In a homogeneous cluster this problem can be solved by the next Theorem

Theorem 9. *Let $\lambda = \frac{r+w}{h}$, and $\bar{p} = \lfloor \lambda \rfloor$. Assume that $\lambda H > R + W$, let i be a positive integer and T a positive real number. The solutions of PUMP is the set of pairs (i, T_i) where $i = 1, ..., \bar{p}$ and*

$$T_i = \frac{i(\lambda H - (R + W)}{\lambda - i}. \tag{4.45}$$

Proof Consequence direct of Theorem 5 proof.□

The values $i = 1, ..., \bar{p}$ induce the partition of the domain $T \geq 0$

$$\{(T_{i-1},\ T_i] :\ i = 1, ..., \bar{p}\} \cup \{(T_{\bar{p}}, +\infty)\} \tag{4.46}$$

where $T_0 = 0$, $(T_{i-1}, T_i] = \{T :\ T_{i-1} < T \leq T_i\}$ and T_i is as above, the solution of $R(i, T) = T$.

4.3.2 Optimal utilization-production over an heterogeneous cluster

The effect induced by a small period is similar to the one induced over the problem of searching the optimal solution to the MP problem in section 4.1.3. This section presents the solution to perfect utilization problem in a special case.

Theorem 10. *Let i be a positive integer, T a positive real number and a fixed activation order. Let $\bar{p}$ be the largest index so that $\sum_{j=1}^{q} \frac{h_j}{r_j+w_j} \leq 1$. The solution of PUMP is the set of pairs* $\textsc{Pump}_{sol}$=$\{(i, T_i) | \sum_{j=1}^{i} H_i - \sum_{j=1}^{i} \frac{R_i+W_i}{r_i+w_i} \geq 0$ and $i = 1, ..., \bar{p}\}$ *and*

$$T_i = \frac{\sum_{j=1}^{i} H_i - \sum_{j=1}^{i} \frac{R_i+W_i}{r_i+w_i}}{1 - \sum_{j=1}^{i} \frac{h_j}{r_j+w_j}}. \tag{4.47}$$

Proof. Let $x_j = (r+w)^{-1}(T)$ the solution given by the approximate solution to MP problem, them

$$R(i,T) = \sum_{j=1}^{i} h\left(\frac{T-(R_j+W_j)}{r_j+w_j}\right). \tag{4.48}$$

Therefore, $R(i,T) = T$ has a solution

$$T_i = \frac{\sum_{j=1}^{i} H_i - \sum_{j=1}^{i} \frac{R_i+W_i}{r_i+w_i}}{1-\sum_{j=1}^{i} \frac{h_j}{r_j+w_j}}, \tag{4.49}$$

and this solution is nonnegative for $i \leq \bar{p}$ such that $\sum_{j=1}^{i} H_i - \sum_{j=1}^{i} \frac{R_i+W_i}{r_i+w_i} \geq 0$.□

As in section 4.3.1 it is possible to find other solutions. These additional solutions occur for values $i > \bar{p}$ where $\sum_{j=1}^{i} H_i - \sum_{j=1}^{i} \frac{R_j+W_j}{r_j+w_j} < 0$. These values of T are closed to 0 and it is possible to get negative values in $\bar{x}$. Similarly, the values $i = 1, ..., \bar{p}$ induce the partition of the domain $T \geq 0$

$$\{(T_j,\ T_i]:\ (i,T_i) \in \textsc{Pump}_{sol}\} \cup \{(T_{\bar{p}}, +\infty)\} \tag{4.50}$$

where $T_0 = 0$, $(T_j, T_i] = \{T:\ T_j < T \leq T_i\}$, j and i consecutive in the $\textsc{Pump}_{sol}\}$ and T_i is as above, the solution of $R(i,T) = T$.

CHAPTER 5
A NEW SCHEDULER FOR A CLUSTER OF WORKERS

5.1 Adjusting the last round

In the literature is common to find that in an optimal schedule, all workers finish at the same time [20]. In [18] a postmortem last round modification is proposed for the UMR algoritm. Here this technique is used to develop a new last round modification method. The new method differs in that, due to the limitations imposed by the period over the rounds, the last round is split in two new rounds, called last round and auxiliary round. Figure 3–3 depicts a scheduler with a last round modification.

5.1.1 The last round in heterogeneous cluster

Modifications of the last round are used either to impose the condition that all processors end operating at the same time or to orchestrate the return of results from the workers to the master. This section presents a general statement of the last round modification problem and a solution. The problem statement includes both, single and multiple port communication networks. Data returns from the workers to the master are considered to be constant in time, and are modeled with the additional affine mappings:

$$s_{i,0}(x) = x.s_{i,0} + S_{i,0}, \text{ send operation from worker } i \text{ to the master} \quad (5.1)$$

$$l_{i,0}(x) = x.l_{i,0} + L_{i,0}, \text{ transmission time from } i \text{ to the master, and} \quad (5.2)$$

$$r_0(x) = x.r_0 + R_0, \text{ receive operation in the master.} \quad (5.3)$$

The modifications alter the values of the (T, k, MG)-partition obtained in Theorem 4 in the last round of the computation. Ideally, the concurrent processes $i = 1, ..., q$ end in the same order that they were started. The delay between any two consecutive workers at the end is simply

$$D_i = h_i(Y_i) \tag{5.4}$$

where $h = e + s$ or $h = l$ as above. It is also assumed that

$$r_0(x) = x.r_0 + R_0 = K \tag{5.5}$$

is the constant time spent in returning one result from a worker to the master. These delays at the end can be compensated by computing an auxiliary $q + 1$-tuple of real numbers, denoted $(Z_1, ..., Z_p, Z_{q+1})$, that modifies the values of the (T, k, MG)-partition $(Y_1, Y_2, ..., Y_{q+1}, 0, ..., 0)$ of X obtained with Theorem 14. The modification alters the amount of core tasks that a processor i computes in two different ways, depending on whether $i \leq m$ or $m < i$, for some index m, $1 \leq m \leq q + 1$ as shown in the Figure 3–3. The modified values are:

$$\text{(a)} \quad Y_i + Z_i, \text{ for } 1 \leq i \leq m \text{ and} \tag{5.6}$$

$$\text{(b)} \quad Z_i, \text{ for } m < i \leq q + 1. \tag{5.7}$$

The new amounts of agglomerated core tasks must satisfy:

$$\sum_{i=1}^{m} (Y_i + Z_i) + \sum_{i=m+1}^{q} Z_i + \chi_{q+1} Z_{q+1} = qY + Y_{q+1}; \tag{5.8}$$

where

$$\chi_{q+1} = \begin{cases} 1, \text{ if } Y_{q+1} \neq 0, \text{ and} \\ 0, \text{ otherwise} \end{cases} \tag{5.9}$$

The tuple $(Z_1, ..., Z_{q+1})$ and the index m are obtained from the following recursive equations:

$$D_i - K + (r_i + w_i)(Z_i) = (r_{i-1} + w_{i-1})(Z_{i-1}),\ 1 < i \leq m; \tag{5.10}$$

$$D_i - K + (r_i + w_i)(Z_{m+1}) = (r_i + w_i)(Y) + (r + w)(Z_m);\ \text{and} \tag{5.11}$$

$$D_i - K + (r_i + w_i)(Z_i) = (r_i + w_i)(Z_{i-1}),\ i > m + 1. \tag{5.12}$$

The following equations express the values Z_i, $i = 2, ..., q + 1$ in terms of Z_1,

$$(r_i + w_i)(Z_i) = \begin{cases} (r_1 + w_1)(Z_1) - \sum_{k=2}^{i} D_x & \text{for } 1 \leq i \leq m \\ (r_i + w_i)(Y_i) + (r_1 + w_1)(Z_1) - \sum_{k=2}^{i} D_x & \text{for } m + 1 \leq i \leq q + 1 \end{cases} \tag{5.13}$$

From the restriction 5.8 it follows that

$$\sum_{i=1}^{q+\chi_{q+1}} Z_i = \sum_{i=1}^{q+\chi_{q+1}} \frac{(r_1 + w_1)(Z_1)}{r_i + w_i} - \sum_{i=2}^{q+\chi_{q+1}} \sum_{j=2}^{i} \frac{(D_j - K)}{r_i + w_i} - \sum_{i=1}^{m} \frac{R_i + W_i}{r_i + w_i} + \sum_{i=m+1}^{\chi_{q+1}} Y_i \tag{5.14}$$

Thus,

$$Z_1 = \frac{\sum_{i=2}^{q+\chi_{q+1}} \sum_{j=2}^{i} \frac{(D_j - K)}{r_i + w_i} + \sum_{i=1}^{m} \frac{R_i + W_i}{r_i + w_i}}{(r_1 + w_1) \sum_{i=1}^{q+\chi_{q+1}} \frac{1}{r_i + w_i}} - \frac{R_1 + W_1}{r_1 + w_1} \tag{5.15}$$

Regarding the computation of m. It is worth remarking that $Z_m \geq 0$ implies $Z_i \geq 0$ for $i = 1, ..., q + 1$, then m is the largest such that $Z_m \geq 0$. This value is found by exhaustive search starting from $m = 1$. The values for Z_i $1 < i \leq q+1$ can be calculated form expression 5.13. A natural way to implement these modification in a SPMD program is splitting the last round into two separate rounds. In the first, processes $i = 1, ..., m$ consist of Y agglomerated core tasks, while processes $i = m + 1, ..., q + 1$ consist of Z_i agglomerated tasks. In the second round, only processes $i = 1, ..., m$ are active, each consisting of Z_i agglomerated tasks. This second round is termed auxiliary round.

5.1.2 Some considerations in homogeneous cluster

In an *Homogeneous Cluster* such as the ones described in section 3.1, let $D = h(Y)$ and assume that the solution $(Y, Y, ..., Y_{q+1}, 0, ..., 0)$ was obtained with Theorem 4. The Z_1 in equation 5.15 can be rewritten as:

$$Z_1 = \frac{R+W}{r+w}(q-m+\chi_{Y_{q+1}}) + \frac{(D-K)(q^2+2q\chi_{Y_{q+1}}-q)}{2(r+w)} - (\chi_{Y_{q+1}}Y - Y_{q+1}). \quad (5.16)$$

The computation of m can be express in close form:

$$Z_m = \frac{(q-m)Y + Y_{q+1}}{(q+1)} - \frac{(q-m+1)(r+w)(Y) + \frac{q(q+1)}{2}(D-K)}{(q+1)(r+w)} \quad (5.17)$$

$$- \frac{(m-1)(D-K)}{r+w} \quad (5.18)$$

For $Z_m \geq 0$ m must be taken to be:

$$m = \left\lfloor \frac{(q+1)(q+2)(D-K) - 2(r+w)(Y-Y_{q+1}) - 2(q+1)(R+W)}{2(q+1)(D-K) + 2(R+W)} \right\rfloor \quad (5.19)$$

If $Y_{q+1} = 0$, let $q = q-1$ and $Y_{q+1} = Y$ in (5.18) and solve similarly.

In order to estimate the make-span of the auxiliary round, the formula of Z_1 is written as:

$$\begin{aligned} Z_1 = & \frac{q(q+1)h - 2(r+w)}{2(q+1)(r+w)}Y + \frac{Y_{q+1}}{q+1} \\ & + \frac{2m(R+W) + q(q+1)(H-K) - 2(q+1)(R+W)}{2(q+1)(r+w)} \end{aligned} \quad (5.20)$$

Since $Y_{q+1} \leq Y$ the value of Y_{q+1} can be absorbed by first term in equation (5.15), to get

$$Z_1 \leq \phi(q)Y + \varphi(q) \text{ where} \quad (5.21)$$

$$\phi(q) = \frac{q(h)}{2(r+w)} \text{ and} \quad (5.22)$$

$$\varphi(q) = \frac{2m(R+W) + q(q+1)(H-K) - 2(q+1)(R+W)}{2(q+1)(r+w)} \quad (5.23)$$

The term (5.22) dominates in equation (5.21). Let $\lambda = \frac{r+w}{h}$ be the ratio between the the time the master spends in preparing and sending a core task, and the time that average worker spends receiving and computing the same task. This ratio is an upper limit for the value of q obtained in Theorem 4. Then,

$$\phi(q) = \frac{q}{2\lambda}. \tag{5.24}$$

After a few algebraic manipulations of equation (5.19), it is concluded that $m \leq \frac{q+2}{2}$. Since $q \leq \lambda$,

$$\phi(q) \leq \frac{1}{2} \text{ and } \varphi(p) \leq \frac{\lambda H}{2(r+w)}. \tag{5.25}$$

The previous arguments prove the following theorem:

Theorem 11. *Let $\lambda = \frac{r+w}{h}$. Then, the auxiliary round takes at most*

$$T' = \frac{1}{2}T + \frac{\lambda H}{2} \tag{5.26}$$

units of time.

This theorem can be used to approximate the reduction in time in the modification of the last round. In the next section it is also uses as an upper bound with a small modification. In heterogeneous systems, the H is replace with H_α=max$\{H_i | 1 \leq i \leq \bar{p}\}$

5.2 Make-span minimization

The final objective of the proposed schedule is to minimize the total execution time. The next mapping model the total execution time as a function the number of rounds n and the period T.

$$\mu(n,T) = (n-1)T + (e + g_1 + r_1 + w_1)(Y_1) + \frac{1}{2}T + \frac{\lambda H_\alpha}{2}. \tag{5.27}$$

Here $(e + g_1 + r_1 + w_1)(Y_1)$ corresponds to the time spent by the first worker in completing the first round of computations. The last term corresponds to the approximation described in section 5.1.2. Since Y_1 depends on T, mapping (5.27)

is well-defined. The actual number n is obtained after the discretization process discussed in section 5.3.

5.2.1 Make-span minimization constrained to maximal production

This section assumes that p, the number of workers, is large enough so $\bar{p} < p$ and that $\nu > 1$ is a real number. The Make-span minimization problem (MM-MP) constrained by the approximation to Maximal Production problem develop in section 4.1.3 is stated as follows:

$$\begin{aligned}
\text{Minimize } \mu(\nu, T) &= (\nu - 1)T + (e + g_1 + r_i + w_i)(Y_1) + \frac{1}{2}T + \frac{\lambda H_\alpha}{2} && (5.28)\\
\text{Subject to } X &= \nu \sum_{i=1}^{q} Y_i + Y_{q+1} && (5.29)\\
T &= (r_i + w_i)(Y_i), 1 \leq i \leq q && (5.30)\\
T &= \sum_{i=1}^{q+1} h_i(Y_i), \text{ and} && (5.31)\\
T &\geq (r_{q+1} + w_{q+1})(Y_{q+1}) && (5.32)
\end{aligned}$$

Since $\frac{\lambda H_\alpha}{2}$ is constant, it is eliminated from the minimization problem. Besides, the analysis will be restricted to values of T within the bounded intervals in partition (4.50). This in turn, eliminates restriction (5.32) and allows for the reformulation of the problem as a set of problems, one per each bounded interval $(T_{i-1},\ T_i]$. A problem in this set is described for a fixed i, as: "$\mathscr{P}_i$:

$$\begin{aligned}
\text{Minimize } \mu(\nu, T) &= (\nu + \frac{1}{2})T + (e + g_1)(Y_1) && (5.33)\\
\text{Subject to } X &= \nu \sum_{j=1}^{i} Y_i + Y_{i+1} && (5.34)\\
T &= (r_j + w_j)(Y_j), 1 \leq j \leq i && (5.35)\\
T &= \sum_{j=1}^{i+1} h_i(Y_i)\text{"} && (5.36)
\end{aligned}$$

There are also $\bar{p}$ problems in this set. After each problem is solved, the subset of values of T satisfying the equation (5.32) with $i = q$ is selected. The minimal value of $\mu(T, Y_1)$ for the T's ranging over the latter subset is the solution to MM-MP.

Definition 10. *The worker that achieves minimal make-span in the latter set is labeled* p_{opt}.

The next theorem is the main result in this section.

Theorem 12. *Let* X *be a nonnegative real number and* i *the number of processors used by* $(T_{i-1}, T_i]$ *fixed interval in (4.50). Then the solution to problem* $\mathscr{P}_i$ *is,*

$$T = \frac{1}{\mathrm{a}(i)} \sqrt{\frac{b(i)(r_1 + w_1)X}{\frac{1}{2}(r_1 + w_1) + e + g_1}} + \frac{b(i)}{\mathrm{a}(i)} \tag{5.37}$$

where $\mathrm{a}(i) = \sum_{j=1}^{i+1} \frac{1}{r_j + w_j} + \frac{1}{h_{i+1}}(1 - \sum_{j=1}^{i+1} \frac{h_j}{r_j + w_j})$ *and* $b(i) = \sum_{j=1}^{i+1} \frac{R_j + W_j}{r_j + w_j} + \frac{1}{h_{i+1}}(\sum_{j=1}^{i+1} H_i - \sum_{j=1}^{i+1} \frac{h_j(R_j + W_j)}{r_j + w_j})$.

Proof The problem is solved by using Lagrange multipliers [48]. First let use 5.35 and 5.36 to conclude that

$$\sum_{j=1}^{i} Y_i + Y_{i+1} = \mathrm{a}(i)T - b(i) \tag{5.38}$$

where $\mathrm{a}(i) = \sum_{j=1}^{i+1} \frac{1}{r_j + w_j} + \frac{1}{h_{i+1}}(1 - \sum_{j=1}^{i+1} \frac{h_j}{r_j + w_j})$ and $b(i) = \sum_{j=1}^{i+1} \frac{R_j + W_j}{r_j + w_j} + \frac{1}{h_{i+1}}(\sum_{j=1}^{i+1} H_i - \sum_{j=1}^{i+1} \frac{h_j(R_j + W_j)}{r_j + w_j})$.

The Lagrange multiplier is:

$$L(\nu, T, \lambda) = \mu(\nu, T) + \lambda\, G(\nu, T). \tag{5.39}$$

where

$\mu(\nu, T) = \left(\nu + \frac{1}{2}\right) T + (e + g)((r_1 + w_1)^{-1}(T))$

$G(\nu, T) = \nu(\mathrm{a}(i)T - b(i)) - X = 0$

Lagrange system is :

$$\begin{cases} \frac{\partial L}{\partial \lambda} = G = 0 \\ \frac{\partial L}{\partial \nu} = \frac{\partial \mu}{\partial \nu} + \lambda \frac{\partial G}{\partial \nu} = 0 \\ \frac{\partial L}{\partial T} = \frac{\partial \mu}{\partial T} + \lambda \frac{\partial G}{\partial T} = 0 \end{cases} \tag{5.40}$$

which is, in turn, equivalent to solve:

$$a(i)T - b(i) = \frac{X}{n} \tag{5.41}$$

$$T + \lambda(a(i)T - b(i)) = 0 \tag{5.42}$$

$$(\nu + \frac{1}{2})(r + w) + (e + g) + \nu a(i) = 0 \tag{5.43}$$

Now, solving for ν and λ in 5.41 and 5.42 yields

$$\lambda = \frac{-T}{a(i)T - b(i)} \text{ and } \nu = \frac{X}{a(i)T - b(i)} \tag{5.44}$$

By substituting in (5.43), it follows that

$$\left(\frac{1}{2}(r_1 + w_1) + e + g_1\right)(a(i)T - b(i))^2 - b(i)(r_1 + w_1)X = 0 \tag{5.45}$$

Finally, by clearing T,

$$T = \frac{1}{a(i)}\sqrt{\frac{b(i)(r_1 + w_1)X}{\frac{1}{2}(r_1 + w_1) + e + g_1}} + \frac{b(i)}{a(i)} \tag{5.46}$$

this conclude the proof.□

As remarked previously, there is no guarantee that T in (5.46) is indeed a member of the ith bounded interval. As a matter of fact, only values i and T satisfying (5.32) contain potential solutions to MM-MP. Using (5.32) and Theorem 12, MM-MP is solved by finding the minimal value of $\mu(T)$ for the periods T's ranging over the subset of values satisfying

$$(r_{q+1} + w_{q+1})(Y_{q+1}) \leq T \tag{5.47}$$

This equation leads to a linear time search algorithm for computing p_{opt}. A natural upper bound for i is $\bar{p}$, the largest index so that $\sum_{j=1}^{q} \frac{h_j}{r_j+w_j} \leq 1$ or in homogeneous case $\bar{p} = \lfloor \frac{A^r+A^w}{A^h} \rfloor$.

5.2.2 Make-span minimization constrained to perfect worker's utilization

It is assumed now that $p \leq p_{opt}$. The problem of make-span minimization under perfect worker's utilization (MM-PWU) is stated as:

$$\text{Minimize } \mu(T) = (\nu + \frac{1}{2})T + (e + g_1)(Y_1) \tag{5.48}$$

$$\text{Subject to } \nu \sum_{i=1}^{p} Y_i = X \tag{5.49}$$

$$(r_i + w_i)(Y_i) = T, 1 \leq i \leq p \tag{5.50}$$

$$\sum_{i=1}^{p} h_i(Y_i) \leq T \tag{5.51}$$

Following procedures that are similar to those outlined in the previous subsection, a set of problems S_i is built for each $i = 1, ..., \bar{p}$. Thus, "S_i :

$$\text{Minimize } \mu(T) = (\nu + \frac{1}{2})T + (e + g)(Y) \tag{5.52}$$

$$\text{Subject to } \nu \sum_{i=1}^{p} Y_i = X \tag{5.53}$$

$$(r_i + w_i)(Y_i) = T, 1 \leq i \leq p\text{"} \tag{5.54}$$

Theorem 13. *Let X be a nonnegative real number and i the number of processors used by $(T_{i-1}, T_i]$ fixed interval in (4.50). Then, the solution to S_i is*

$$T_{sol}^{(i)} = \frac{1}{a(i)} \sqrt{\frac{b(i)(r_1 + w_1)X}{\frac{1}{2}(r_1 + w_1) + e + g_1}} + \frac{b(i)}{a(i)} \tag{5.55}$$

where $a(i) = \sum_{j=1}^{i} \frac{1}{r_j+w_j}$ *and* $b(i) = \sum_{j=1}^{i} \frac{R_j+W_j}{r_j+w_j}$.

Proof. Similar to theorem 12.

Similar to the previously problem, there is no guarantee that $T_{sol}^{(i)}$ in (5.55) is indeed a member of the ith bounded interval. As a matter of fact, only values

i and T satisfying (5.51) contain potential solutions to MM-MWU. Using (5.51) and Theorem 13, MM-MWU is solved by finding the minimal value of $\mu(T)$ for the periods $T_{sol}^{(i)}$ ranging over the subset of values satisfying

$$T_{sol}^{(i)}(a(i)-1) \leq b(i) \tag{5.56}$$

If only a limited subgroup $i < \bar{p}$ of workers is used, the solution to the MM-PWU problem can be founded by

$$T = \begin{cases} \arg\min \{\mu(T_{sol}^{(i)}), \mu(T_{i-1}), \mu(T_i)\} \text{ if } T_{sol}^{(i)} \in (T_{i-1}, T_i] \\ \arg\min \{\mu(T_{i-1}), \mu(T_i)\} \quad \text{otherwise} \end{cases} \tag{5.57}$$

in this solution if the minimum is achieved in $T = T_{i-1}$ then, the solution only uses $i-1$ workers.

5.3 Discretization in homogeneous cluster

So far, all parameters except p are real numbers. The SCOW parameters `nRounds` and `nCoreTasks` are obtained by discretizing the real parameters. The discretization process is carefully crafted to preserve the characteristics of maximal production or perfect utilization per round of the continuous parameters.

5.3.1 Discretize the number of agglomerated tasks

There are two main options when it comes to discretizing parameter Y: take either its floor or its ceil. Either value may be the optimal number of core tasks for the SPMD code. Thus,

$$\texttt{nCoreTasks} = Y^E = \lfloor Y \rfloor \text{ or } \lceil Y \rceil . \tag{5.58}$$

Each choice induces in turn, a different value for the period T through

$$T^E = (r+w)(Y^E); \tag{5.59}$$

and for the delay parameter D that is used for the adjustment of the last round, via

$$D = h(Y^E). \tag{5.60}$$

It follows that

$$\texttt{nWorkers} = q = \left\lfloor \frac{T}{D} \right\rfloor. \tag{5.61}$$

These values, provide in turn $T_\epsilon = T - qD$, and with it

$$Y^E_{q+1} = \left\lfloor h^{-1}(T_\epsilon) \right\rfloor. \tag{5.62}$$

The number of rounds is thus,

$$\texttt{nRounds} = N^E = \left\lfloor \frac{X}{qY^E + Y^E_{q+1}} \right\rfloor \tag{5.63}$$

With these parameters, each period performs

$$\text{Tasks per period} = qY^E + Y^E_{q+1}, \tag{5.64}$$

and therefore, the total amount of work covered is

$$N^E(qY^E + Y^E_{q+1}). \tag{5.65}$$

The remaining work $X_\epsilon = X - N^E(qY^E + Y^E_{q+1})$ is added to the last round. The way this remaining amount of work is added depends upon two additional auxiliary parameters. These are

$$a = \frac{2qw - (q+1)(q+2)h}{2w} \text{ and } b = \frac{2W - (q+1)(q+2)H}{2w}. \tag{5.66}$$

If $X_\epsilon > aY^E + b$ then X_ϵ tasks are enough to complete an additional round. In this case, equation 5.8, is to be replaced with

$$\sum_{i=1}^{m} (Y + Z_i) + \sum_{i=m+1}^{q} Z_i + \chi_{Y_{q+1}} Z_{q+1} = X_\epsilon. \tag{5.67}$$

Otherwise, the X_ϵ tasks are added in the last round together with the adjustment described in section 5.1. With these modifications, equation (5.8) becomes

$$\sum_{i=1}^{m}(Y+Z_i)+\sum_{i=m+1}^{q}Z_i+\chi_{Y_{q+1}}Z_{q+1}=qY^E+Y_{q+1}^E+X_\epsilon. \tag{5.68}$$

Because of the previous discussion the schedule has two possible make-spans. These are,

$$\text{Make}-\text{span} \;=\; \begin{cases}(e+g)(Y^E)+(N^E+1)T+(r+w)(Z_1), \text{ if } X_\epsilon > aY^E+b \\ (e+g)(Y^E)+N^ET+(r+w)(Z_1); \text{ otherwise.}\end{cases}. \tag{5.69}$$

Since Y^E has two possible values, the solution is the minimum make-span between these two vales.

5.3.2 Discretization the number of rounds

The previous procedure discretizes the number of agglomerated tasks per round. An alternative is to discretize the number of rounds necessary to perform the whole task. In this case, there are two possible values for

$$\texttt{nRounds} \;= N^E = \lfloor \nu \rfloor \text{ or } \lceil \nu \rceil . \tag{5.70}$$

Each value provides, in turn, an estimation for

$$\text{Total work in a round} \;= \frac{X}{N^E} = qY + Y_{q+1}. \tag{5.71}$$

Thus, the problem is finding q and Y such that

$$T \;=\; (r+w)(Y) = qh(Y)+h(Y_{q+1}); \text{ and} \tag{5.72}$$

$$Y_{q+1} \;<\; Y. \tag{5.73}$$

In this case, the value of Y is not discretized. The next procedure finds the values of q and Y for a given N^E.

For i=1:$\lfloor\lambda\rfloor$

$T = h(\frac{X}{N^E}) + (i+1)H$;

$Y = (r+w)^{-1}(T)$;

if $i = \left\lfloor \frac{T}{h(Y)} \right\rfloor$

$q \leftarrow i$;

end

end

5.3.3 Implementation issues

There are several consideration related to the implementation of SCOW. Initially the input values to SCOW are the values obtained by a simple linear regression method, over a minimal set of measurements. Due to the error in the regression, the accuracy of the predictions is a difficult problem. In the section 3.1.3 Figure 3–5 shows two different areas called affine windows. If the first window is use to predict the communication time, and if x takes values (> 450), the approximation error is very large. This problem induces an additional restriction. The values of x are to be kept within the affine windows determined by the regression.

Another important issue is that the parameters estimated by regression may not be consistent with the theory. For example, the values of overhead produced by regression may be negative in some situations. In these cases, the model loses validity and return wrong optimal values. In the regression obtained by the values in the second window in Figure 3–5, the estimated parameters returned for the send a receive mappings have indeed negative overheads. This is shown in Table 6–7. In this particular case, the problem can bypassed because the model is able to absorb negatives overheads in the compute mapping in such a way that the overhead parameter of mapping h result ultimately positive.

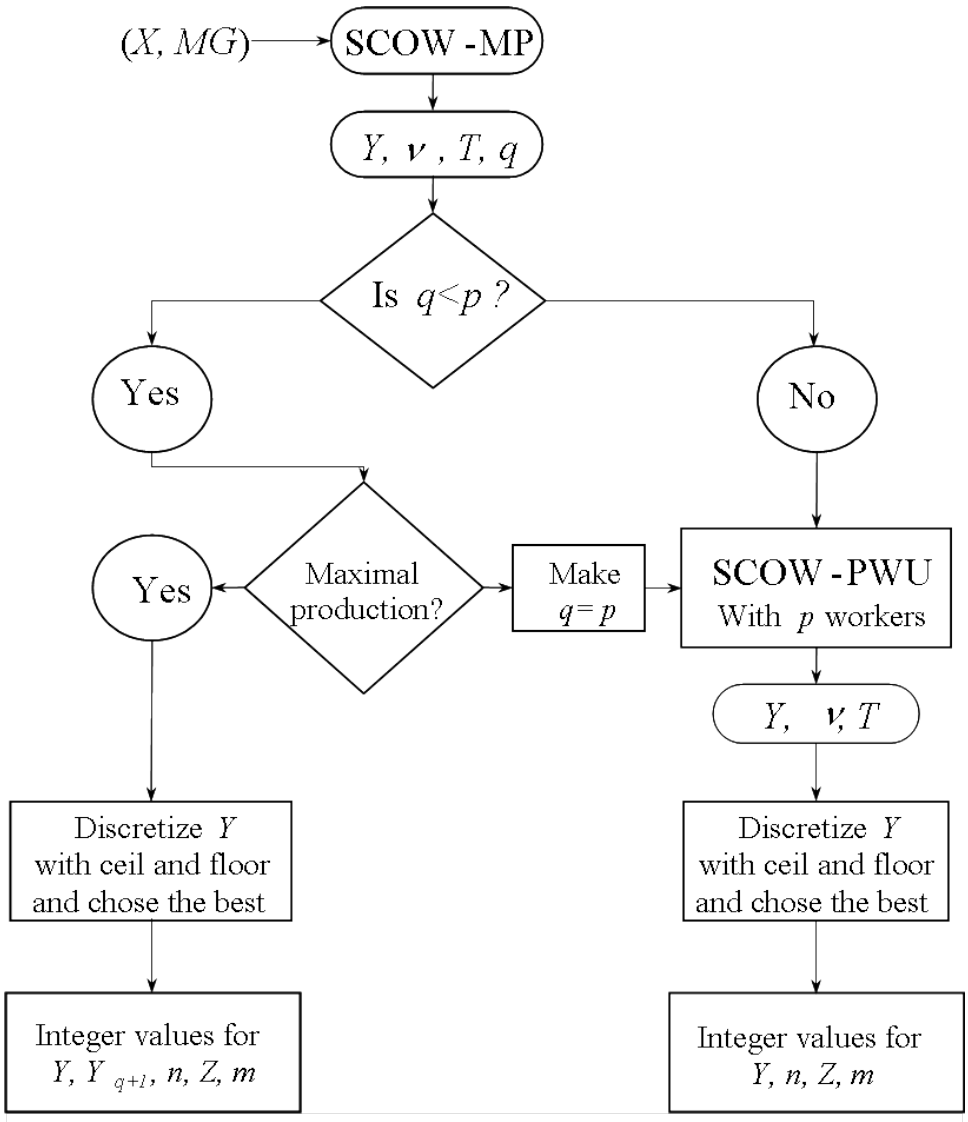

Figure 5–1: Flow-Chart for the Computation of SCOW Parameters

5.3.4 Implementing SCOW in a SPMD code segment

The whole computation of the SCOW parameters are described in the flowchart depicted in Figure 5–1. The user provides a divisible task with a well-defined core task. The basic performance parameters derived from a few measurements of the core is the following MG

$$MG = \{(OP,\ op) \mid op(x) = x.op + OP,\ op = e,\ s,\ l,\ r,\ \text{or}\ w,\} \tag{5.74}$$

These MG and the total amount of core tasks X are fed into the mathematical equations of the continuous model. This is done in node SCOW-MP in Figure 5–1, which corresponds to the process implemented in the MATLAB function in Figure 5–2. The output of SCOW-MP are the number of agglomerated tasks, Y; a real approximation to the number of rounds, ν; the length of the period, T; and the optimal number of workers required, this is q. It is also assumed that $p > 1$ workers are available in the cluster. Now, if $q < p$, the user may opt for either minimizing the make-span with maximal production, or minimizing the make-span under the perfect worker's utilization constraint. Otherwise, if $q \geq p$, the best option is clearly to minimize make-span under perfect worker's utilization. After these decisions are made, the discretization procedures discussed in section 5.3 are applied to get the actual SCOW parameters.

```
function Values = scow_mp(X,MG)
E=MG(1,1);H=MG(2,1);G=MG(3,1);R=MG(4,1);W=MG(5,1);
e=MG(1,2);h=MG(2,2);g=MG(3,2);r=MG(4,2);w=MG(5,2);
bestu=w*X;
lamda=(r+w)/h;
for i=1:1:floor(lamda)
   T=sqrt((X*h*(r+w)*(i+1)*H)/(e+g+1/2*(r+w)))+(i+1)*H;
   Y=(T-(R+W))/(r+w);
   HY=h*Y+H
   if floor(T/HY)==i
       v=X*h/(T-(i+1)*H);
       U=(v+1/2)*T+(e+g)*Y+E+G;
       if U<bestu
           bestu=U;
           Values=[Y,v,T,i];
       end
   end
end
```

Figure 5–2: Matlab Implementation for the Continuous Model

CHAPTER 6
EXPERIMENTAL RESULTS

This chapter is organized as follow: in section 6.1 the results of numerical comparisons of SCOW with others schedulers is shown. The time executions of two parallel implementations, one scheduled with SCOW and the other using a fist-in first-out job distribution, of a Bioinformatics optimization algorithm is presented in section 6.3. In the section 6.4.1 SCOW and UMR are used to schedule the above mentioned Bioinformatics algorithm. Finally, a parallel branch and bound implementation of the same Bioinformatics problem is presented in 6.5.

6.1 Numerical comparison

Table 6–1: Simulation Parameters

Parameter	Values
Agglomerated tasks	$X = 1000$
Computational rate	$S = 1; r = 0.1\frac{1}{S}; w = 0.9\frac{1}{S}$
Transfer rates	$B = 1.1 \times N, 1.1 \times N + 1; l = \frac{1}{B}$, where N=10,20,30
Linear constant of link operation	$l = \frac{1}{B}$
Computational latency	$cLat = 0.03; R = 0.1cLat; W = 0.9cLat;$
Communication latency	$nLat = 0.03; L = nLat;$
Retrieval and send	$E = 0; e = 0; S = 0; s = 0;$
Auxiliary mapping	$g = s + l$ and $h = l$

Some UMR numerically predicted performance values are presented in [45]. A few of these values, which were randomly selected, are used in this section for predicting the corresponding SCOW-MP and SCOW-PWU performance values. These predictions are made using solely the mathematical equations underlying them. It

is worth remarking that UMR methods find the optimal number of rounds, and perform no discretization on the amount of work per round. Thus, in order to make the comparisons possible, SCOW discretizations are made on the number of rounds and not on the amount of agglomerated tasks per round. The randomly chosen values are shown in Table 6–1.

The numerical prediction of the performances of UMR and SCOW are shown in Table 6–2. In this table, each cell is organize by: (1,1) make-span, (1,2) number of workers, (2,1) workers' utilization and (2,2) network's utilization.

Table 6–2: Numerical Experimental Results of SCOW and Competing Methods

N	B	SCOW-MP		PERIODIC		UMR2		SCOW-PWU		UMR2		UMR	
10	11	99.058	11	99.198	11	99.181	11	102.685	10	=UMR		102.528	10
		0.962	1	0.951	1	0.960	1	1	0.998			1	0.992
20	22	53.427	21	53.613	22	55.489	20	55.176	19	56.946	19	57.567	18
		0.971	1	0.905	1	0.974	1	1	0.997	0.993	1	1	0.991
30	33	38.236	30	38.795	33	39.668	30	38.248	29	40.144	29	41.298	25
		0.986	1	0.8457	1	0.941	1	0.999	1	0.958	1	1	0.991
10	12	91.453	12	91.535	12	91.591	12	93.670	11	=UMR		93.633	11
		0.958	1	0.948	1	0.956	1	1	0.999			1	0.997
20	23	51.434	22	51.739	23	52.411	22	51.477	21	53.314	21	53.189	20
		0.961	1	0.888	1	0.945	1	0.998	1	0.970	1	1	0.992
30	34	37.321	31	37.495	34	38.705	31	37.345	30	39.143	30	40.483	26
		0.977	1	0.846	1	0.936	1	0.987	1	0.953	1	1	0.995

Table 6–3 shows the normalized make-span and utilization of SCOW versus the competing schedulers, this is PERIODIC, UMR and UMR2. By normalization it is understood the following metric

$$\frac{1}{\eta}\sum_{i=1}^{\eta}\frac{\Psi \text{ of competing scheduler with simulation } i}{\Psi \text{ of SCOW with simulation } i}, \tag{6.1}$$

Where η is the number of numerical simulations selected for the comparison, and Ψ stands for either make-span or utilization. Because of the characteristics of each of the competing schedulers, not all normalizations yield useful results. For

instance, in several cases UMR does not use the same number of workers as SCOW-MP. Therefore, in those cases, it does not make sense normalizing the make-span as SCOW-MP will always outperform UMR. A similar situation occurs with SCOW-PWU and UMR whenever the number of workers used by each scheduler is different. The only useful normalizations are the ones represented in Table 6–3.

Table 6–3: Comparison Between SCOW and Competing Methods, Average over 36 Experiments

	SCOW-MP	PERIODIC	UMR2	SCOW-PWU	UMR2	SCOW-PWU	UMR
Normalized Make-span	1.000	1.005	1.022	1.00	1.027	1.000	1.000
Normalized Utilization	1.000	1.038	1.008	1.00	1.009	1.000	1.001

The data in the table shows that SCOW outperforms UMR, UMR2 and PERIODIC. SCOW is superior in system utilization. On the other hand, PERIODIC has a better make-span than SCOW-MP in some numerical simulations. This is due to the fact that PERIODIC uses more workers than SCOW-MP. However, this gain in time brings about the poorest utilization in the whole table.

6.2 Comparisons Through Simulations

According to [49] *SimGrid is a toolkit that provides core functionalities for the simulation of distributed applications in heterogeneous distributed environments. The specific goal of the project is to facilitate research in the area of distributed and parallel application scheduling on distributed computing platforms ranging from simple network of workstations to Computational Grids.*

In this section SimGrid is used to validate the numerical predictions made in section 6.1. As shown in Table 6–4, the simulations of SCOW show no significant differences with corresponding predicted values. This was not the case with the simulations of PERIODIC, UMR and UMR2, The main reason behind these differences lies in the last round, which is in most cases modified so that all processors end

Table 6–4: Numerical Experimental Results of SCOW and Competing Methods

N	B	SCOW-MP		PERIODIC		UMR2		SCOW-PWU		UMR2		UMR	
10	11	99.063	11	99.254	11	98.688	11	102.685	10	=UMR		102.528	10
		0.962	1	0.952	1	0.960	1	1	0.998			1	0.992
20	22	53.429	21	53.309	22	53.907	20	55.176	19	54.882	19	57.567	18
		0.971	1	0.905	1	0.974	1	1	0.997	0.993	1	1	0.991
30	33	38.239	30	38.331	33	38.841	30	38.248	29	39.385	29	40.795	25
		0.987	1	0.846	1	0.941	1	0.999	1	0.958	1	1	0.991
10	12	91.457	12	91.536	12	91.202	12	93.670	11	=UMR		93.807	11
		0.958	1	0.949	1	0.956	1	1	0.999			1	0.997
20	23	51.436	22	51.428	23	51.772	22	51.477	21	52.699	21	54.763	20
		0.962	1	0.897	1	0.945	1	0.998	1	0.970	1	1	0.992
30	34	37.323	31	37.009	34	37.991	31	37.345	30	38.519	30	40.567	26
		0.977	1	0.861	1	0.936	1	0.987	1	0.953	1	1	0.995

operating at the same time. In the particular case of PERIODIC, such modification is the same as the one in SCOW. But due to the weak continuity of the workers in PERIODIC, the aim of all workers ending at the same time is seldom achieved. A post mortem improvement on the simulation results is made in Table 6–4. This improvement consisted in re-distributing the last round lengths in such a way that all workers end in the average of their ending times. Unfortunately, no efficient algorithm is available for performing such re-distribution automatically.

Table 6–5: Comparison Between SCOW and Competing Methods using SimGrid.

	SCOW-MP	PERIODIC	UMR2	SCOW-PWU	UMR2	SCOW-PWU	UMR
Normalized Make-span	1.000	0.999	1.007	1.000	1.013	1.000	1.000
Normalized Utilization	1.000	1.036	1.009	1.000	1.010	1.000	1.001

As for UMR, the problem lies in the fact that last round modification provided by the authors fails in some cases. Indeed, whenever the round length increments are sufficiently small there is a possibility that lose continuity between rounds. This is naturally reflected in differences in the processors ending times and therefore, a longer make-span. Finally, in the case of UMR2 no last round modification is

provided by the authors. Nonetheless, the last round modification provided for UMR can be modified for UMR2. Table 6–5 shows the improvements in the UMR2 and PERIODIC performance when these new modification is applied.

6.3 An improved parallel brute force motif finding solver

Given a set of t DNA sequences, the motif finding problem consists in finding a set of t l-mers, this is, strings with l characters; one from each sequence, that maximizes the consensus score. The consensus score is computed by selecting one position in each of the t sequences, thus forming an array $s = (s_1, ..., s_t)$. The l-mers starting at these positions are arranged as a $t \times l$ matrix, called alignment matrix. The alignment matrix is used, in turn, to compute the profile matrix. The latter is a $4 \times l$ array of nonnegative integers. Each row in the profile matrix corresponds to one of the nucleotides A, T, G and C; and each entry (i, j) holds the number of times that nucleotide i appears in column j. The consensus score of a profile matrix with starting positions $s = (s_1, ..., s_t)$ is the sum of the highest scores in each column. This score is denoted $Score[(s_1, ..., s_t)]$. The set of t l-mers with the highest $Score[(s_1, ..., s_t)]$ is the computational solution of the motif finding problem.

Next is the pseudo-code of a brute-force algorithm for solving the motif finding problem.

```
Set BestScore ← 0
For each (s_1, ..., s_t)
    Compute Score[(s_1, ..., s_t)]
    If Score[(s_1, ..., s_t)] > BestScore
       BestScore ← Score[(s_1, ..., s_t)]
       Motif ← (s_1, ..., s_t)
Return Motif.
```

In the parallel motif finding algorithm, the master distributes among the workers packages of alignment matrices. Each worker produces the profiles of the matrices

in its package, computes the score, and keeps best score among all received packages. After the last round, the master receives the scores from the workers and computes the best score. Table 6–6 shows the execution times of the parallel motif finding algorithm scheduled with SCOW against one that implements a first-in,

Table 6–6: Experimental Result of SCOW and FIFO

$t = 6, l = 8$	$m = 20$		$m = 22$	
	FIFO	SCOW-MP	FIFO	SCOW-MP
Observed Time	3.4595	2.8976	39.1750	32.3308
Predicted Time	N/A	2.6832	N/A	30.0358
Workers	26	14	27	14
Y	450	450	450	450

first-out (FIFO) policy for job distribution. In the second case, a extensive search was done to allocate the best possible amount of agglomerated tasks. The table show only the best execution times observed for the FIFO job distribution. The SCOW column shows both, the actual execution time and the execution time predicted by the model. Experiments were run over an IBM cluster with 64 Intel Zeon 2.8 GHz processors, star connected with a bandwidth of 100 Gb per second.

As the Table 6–6 shows, SCOW yields a superior performance both, in terms of time and system utilization.

6.4 Experimental comparison with UMR

Table 6–7: Mapping Regression

Mapping	Regression on [25, 450]	Regression on [500, 2500]
$e(x)$	4.6E-07x+2.2E-05	4.5E-07x+5.7E-06
$s(x)$	3.2E-07x+2.2E-05	4.0E-06x-1.1E-03
$r(x)$	3.9E-07x+2.3E-05	4.1E-06x-2.3E-04
$w(x)$	1.1E-05x+2.0E-05	1.1E-05x-1.0E-04

Figures 6–1 and 6–2, which were obtained from measurements of actual runs of the implementation discussed in section 6.3, confirm (3.5) for x ranging over the intervals $[25, 450]$. These graphs depict the execution time of the master's message

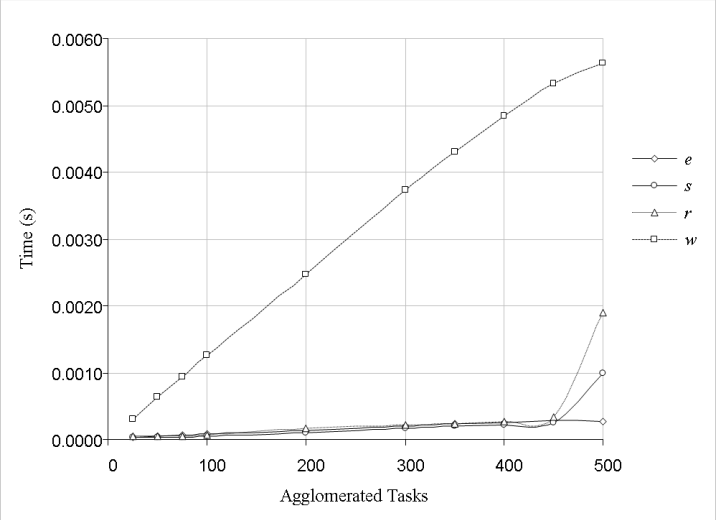

Figure 6–1: The Affine Window [25, 450] for e, s, r and w.

preparation (e) and send tasks (s); and the workers' receive (r) and compute(w) tasks. The symbol l represents the time spent in the transmission of x agglomerated tasks over the network link. This variable appears in network centric computations , this is, computations in which the communications network is saturated. In our experiments, the master processes are the ones that remain saturated in the sense that the master processor has no idle times. As a consequence, l is replaced with $e + s$. Although regression over $[25, 450]$ confirms the affine model, between 450 and 500 linear regression does not render an affine graph. Over the interval $[500, 2500]$, linear regression renders affine mappings again but, as shown in Table 6–7, with a negative *Overheat* for the mappings of send, receive and compute. Thus, the validity of the affine model is limited to what called an affine window. This window depends on the problem and the underlying computing system. In the particular case of the implementation measured in this experiment, the affine window is $[25, 450]$. This affine window is incorporated as a *post-mortem* constrain in the design of SCOW. This means that if the optimal solution of the mathematical model lies outside the affine window, the actual solution is taken to be the one that produces the minimal make span between the extreme values of the window. No considerations to this limitation is made in the UMR scheduler's literature.

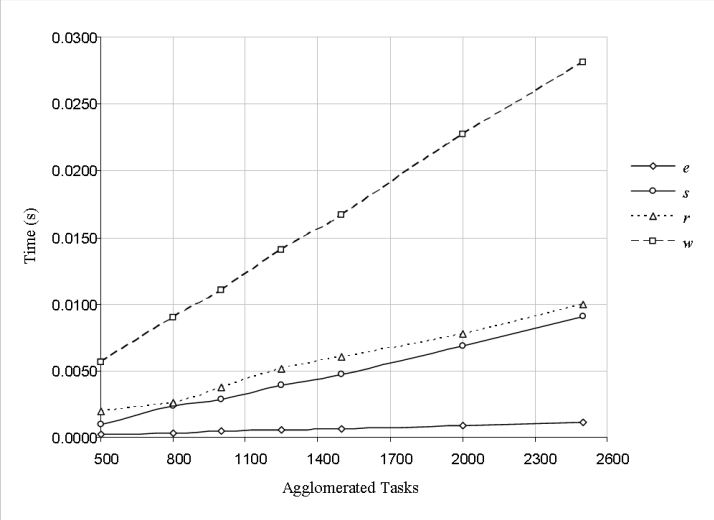

Figure 6–2: The Affine Window [500, 2500] for e, s, r and w.

6.4.1 Scheduling a Motif Finding Solver

As discussed above, the basic measurements taken for the parallel motif finder were: "data retrieval (e)", "MPI send (s)", "MPI receive (r)" and "compute (w)". Since the master processor executes no consensus computations, the following assumptions were made to cast these measurements in terms of the UMR model:

$$Tcomm_i = e + s, \text{ and} \tag{6.2}$$

$$Tcomp_i = r + w. \tag{6.3}$$

The algorithm for computing UMR parameters returns the optimal amount of rounds and the chunk sizes of each round. As pointed out before, although UMR provides a discretized value for the number of rounds, the chunk sizes are still nonnegative real numbers. In order to overcome this limitation, in our experiments, the chunk sizes were rounded and the remaining loads were incorporated into the last round.

The input for the algorithm that computes the parameters of SCOW is the set of values op and OP, with the mappings $op = e$, s, r and w obtained with the above discussed linear regressions. Since SCOW is designed for determining the optimal amount of agglomerated core tasks and optimal number of processors, no further considerations have to be made on this algorithm's outputs. In fact, the

outputs of SCOW are the optimal values for `nRounds`, `nWorkers`, and `nCoreTasks`. The experiments were conducted under the restriction of maximal production per period. In this case, it is sometimes possible to get a different value for `nCoreTasks` for the last worker. The parallel motif finding algorithm that result using these parameters fits perfectly the pseudo-code in section 1.2, except by the last worker, which is to be coded separately.

The experiments were conducted with the problem parameters displayed in the next Table 6–8.

Table 6–8: Experimental Parameter

Parameter	Values
Number of DNA sequences	t=6
Size of searched pattern	l=8
Lengths of DNA sequences	m=13,20,26
Total amount of core tasks	$X = (m - l)^t$
Size of x tasks in byte	$x * t * l * sizeof(char)$

The DNA sequences were randomly generated. A pattern with some minor mutations was inserted in each sequence, to simulate actual DNA data. The fact that t and l remain fixed in each problem instance allows for maintaining a standard group of measurements for the affine mappings, and a well-defined core task. Experiments were run over an IBM cluster with 64 Intel Xeon 2.8 GHz processors, star connected with a bandwidth of 100 Gb per second. All experiments are limited to the affine window $[25, 450]$ of sizes of agglomerated tasks.

6.4.2 Experiments with SCOW

Table 6–9 shows the results of three runs with values $m = 13$, $m = 20$ and $m = 26$, respectively; for the parallel motif finder scheduled with SCOW,

where actual number of workers refers to the number of workers obtained after the post-mortem restriction of the original SCOW values to the affine window. The actual number of agglomerated tasks corresponds in turn, to a similar restriction but

Table 6–9: Experimental Result SCOW

$t = 6, l = 8$	$m = 13$	$m = 20$	$m = 26$
Actual Time	0.0232	2.8976	32.3308
Predicted Time	0.0177	2.3903	26.7541
Predicted number of workers	14	15	16
Actual number of workers	14	14	14
Predicted number of agglomerated tasks	363	4558	15762
Actual number of agglomerated tasks	363	450	450

with the sizes of the agglomerated tasks. In the case of $m = 13$ the time predicted by SCOW is close to the actual execution time. The number of agglomerated tasks falls within the affine window, in this case, as well. However, in the cases of $m = 20$ and $m = 26$, the amount of agglomerated tasks originally returned by the algorithm are outside the affine window. As a result, these sizes were set to 450 agglomerated tasks, which is the local optimal obtained by the previously mentioned post-mortem analysis. As the table shows, the number of agglomerated tasks predicted for $m = 20$ and $m = 26$ is significantly higher than 450. However, when these values are fed into the model to estimate the number of workers, this number turns out to be very low. As a result, the actual execution time of the parallel algorithm is much higher than the one showed in the table for the values restricted to the affine window. This distortion shows that the model fails outside the affine window.

6.4.3 Experiments with UMR

The same experiments were conducted with the parallel tasks scheduled with UMR. The next Table 6–10 shows the results for $m = 13$, $m = 20$, and $m = 26$ for $t = 6$ and $l = 8$, as well. For $m = 13$ the results are quite similar to those obtained with SCOW. These similarities can be explained by the fact the all chunk sizes fall within the affine window. However, this is no longer the case for $m = 20$ and $m = 26$. In fact, in these cases, the chunk sizes fall very far from the affine

window. This fact explains the differences between the predicted and the actual make-span measurements.

Table 6–10: Experimental Result UMR

$t = 6, l = 8$	$m = 13$	$m = 20$	$m = 26$
Actual Time	0.0245	12.9534	146.8055
Predicted Time	0.0181	2.5279	26.8482
Optimal number of workers	11	14	15
Range of chunk size expansion	147-232	739-13535	6296-10543

6.4.4 A Hybrid UMR-SCOW scheduler

On one hand, UMR has the potential for reducing the delay of starting the computations as it distributes smaller loads than those predicted by SCOW in the first rounds. On the other hand, the increase in the chunk size per each round presents the potential for getting chunk size values outside the affine window. A hybrid URM-SCOW scheduler that uses UMR for scheduling the first rounds and then switches to the SCOW scheduler to end the process under the SCOW policy, seems to capture the advantages of both methods. This section illustrates how this hybrid scheduling method would work. Work is underway for developing a mathematical model to assess the actual advantages of this hybrid method.

The next UMR-SCOW hybrid scheduler uses the following policy: initially it uses an amount of workers such that the first chunk sizes fall within the affine window. As soon as the time of the round under UMR is close to the optimal period predicted by SCOW, the scheduler switches to SCOW. The next Table 6–11 shows the results of runs of the previously discussed parallel motif finder with the above described UMR-SCOW hybrid scheduler.

The predicted time in the UMR phase is the time spent in sending 7 rounds of data chunks to 13 workers. At the seventh round, an additional package with 248 agglomerated tasks is sent to worker 14, as a preparation for the transition to the

Table 6–11: Experimental Result Hybrid UMR-SCOW

$t = 6$, m=20 $l = 8$	First phase (UMR)	Second phase (SCOW)
Actual Time	2.9434	
Predicted Time	0.0329	2.6058
Number of rounds	7	486
Number of workers	13	14
Chunk sizes	371-448	450

SCOW policy. Then, SCOW takes over, with 450 agglomerated tasks for 13 workers and 222 agglomerated tasks for worker 14. All other parameters of SCOW are used in this phase, including the modification of the last round for the workers to end operating at the same time.

It is worth noticing that in this experiment, the hybrid UMR-SCOW scheduler renders better performance than the UMR scheduler alone but it does not improve over that of the job scheduled with SCOW. A possible explanation for the superiority of SCOW in this experiment is the fact that SCOW uses 14 processors from the beginning of the computation, while, due to the characteristics of UMR, the UMR-SCOW starts only with 13 processors; and the time gained in the starting-up of the workers is not better, in this case, than the time of performing the whole computation with 14 workers.

6.5 A Parallel Biosequence Motif Discoverer Based on Dynamic Programming

An alternative, which is commonly used, is a serial branch-and-bound method (BBM). BBM searches for the best scores through the branches of a tree. This tree is better described in terms of its sub-trees. A sub-tree of this tree is rooted at level r, $1 \leq r \leq t$, r ranging over the indices of the DNA sequences; and has a selected l-character subsequence as its root. All l-character subsequences in the $r+1$th DNA sequence are the descendents of the selected root. The root of the tree is the empty string which is assumed to be at level $r = 0$. As a consequence, the sequence of

nodes in a branch of this tree is an alignment matrix. BBM start by traversing the tree depth-first, accumulating partial scores until reaching the leaf of the selected branch. Thus, once a leaf is reached, BBM has completed the computation of the score for the corresponding alignment matrix. Then, BBM sets *bestScore* as this matrix's scores and steps one node back in the traversed branch. The idea is to explore the sub-tree rooted at this node in light of the current *bestScore*. Indeed, BBM decides whether is worth computing the score of the alignment matrix that corresponds to the $t-1$ nodes in the original branch and a leaf in a sub-tree. If the BBM rule dictates that the computation is worth making then the method computes the score of the alignment matrix and updates *bestScore*. Otherwise, BBM bypasses the leaf, eliminating thus unnecessary computations. Once the sub-tree has been explored, BBM steps one node back in the branch and repeats the elimination-computation-update routine on each of the sub-trees. The process is carried out until BBM steps all the way up to the root of the search tree and completes the elimination-computation-update routine. At each level, BBM computes the score of the alignment matrix only if

$$optimisticScore = Score(s, i) + (t - i)l < bestScore; \tag{6.4}$$

where $Score(s, i)$ is the partial score of the branch of the indices $s = (s_1, ..., s_i)$, for some $i < t$. Otherwise, BBM bypasses the node.

6.5.1 A branch and bound motif search algorithm

A Branch and Bound search has the potential to significantly lower the problem complexity in some particular cases. The Branch and Bound method is implemented as a search over a t - level tree, which is built in such a way that each node in a path from the root to a leaf represents the alignment matrix of a particular choice of starting positions $(s_1, ..., s_t)$. More precisely, the root is the empty string, the alignment matrices are contained in the leaves, and a parent of a node at level $j+1$

is the first j rows of the alignment matrix corresponding to the starting positions $(s_1, ..., s_j)$. The Branch and Bound method traverses the tree by either one of the operations of (a) move to the next leaf (b) visit all the leaves (c) visit the next node, and (d) bypass the children of a node. The tree is traversed in depth - first mode, starting from the root. By performing a visit to next node, from let's say a node in level $i-1$ to one in level i, a new row l - DNA symbol row is added to the alignment matrix and a partial score $Score(s, i)$ is computed. The fact that new rows are added in each computation of a partial score changes the values of w at each level. In order to account for these changes, the work level i is denoted w_i. In any case, every such computation adds at most l to the previous $Score(s, i-1)$. This allows a bounding criterion: "If all subsequent $(t-i)$ positions $(s_{j+1}, ..., s_t)$ add $(t-i) \times l$ to $Score(s, i)$ and if $Score(s, i) + (t-i) \times l < bestScore$, it does not make any sense to search in vertices of the current sub - tree". So, in these cases, the method bypasses the children nodes. Next is a pseudo code for the Branch and Bound motif search algorithm.

BBM($family$,t,n,l)

s $\leftarrow$ (1,...,1)

bestScore $\leftarrow$ 0

$i \leftarrow 1$

While $i > 0$

if $i < t$

optimisticScore $\leftarrow Score(s, i) + (t-i) \times l$

If optimisticScore $< bestScore$

$(s, i) \leftarrow Bypass(s, i, n-l+1)$

Else

$(s, i) \leftarrow NextVertex(s, i, n-l+1)$

Else

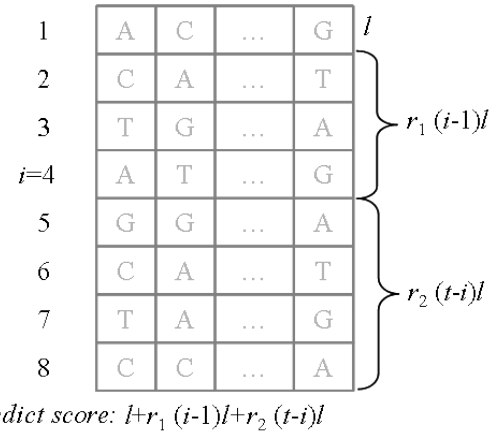

Figure 6–3: Predicted Score Model

If $Score(s) > bestScore$

$bestScore \leftarrow Score(s)$

$bestMotif \leftarrow (s_1, s_2, , s_t)$

$(s, i) \leftarrow NextVertex(s, i, t, n - l + 1)$

Return bestMotif

6.5.2 Analysis of the bounding rule

This section examines the effectiveness of the bounding rule to eliminate unnecessary computations. More precisely, the aim is to establish at what level in the tree the bounding rule is expected to start eliminating a significant number of partial profile matrices. In order to perform this analysis, two approximate fractions, namely r_1 and r_2, are used to model the changes in the partial score. The model is illustrated in Figure 6–3. In mathematical terms, the model predicts a score of the form of

$$predictedscore = l + r_1(i-1)l + r_2(t-i)l \tag{6.5}$$

where r_1 is a parameter for approximating the best value of a partial score, and r_2 represent the average fraction added to complete the predicted score. In order to establish an interval within the partial scores in which the computation of a new score is judged to be unnecessary by the rule 6.4, it is assumed that a generic partial score can be represented as $Score(s, i) = K(i) + \epsilon$. Here $K = \lceil \frac{i}{4} \rceil$

is the minimal partial score value that is always found at each level and $\epsilon \geq 0$. All things considered, the problem reduces to finding level i and $\epsilon \geq 0$ in which $optimisticScore < predictedscore$. This is,

$$K(i) + \epsilon + (t-i)l < l + r_1(i-1)l + r_2(t-i)l \tag{6.6}$$

Statistics taken from experimental data in the problem considered in this dissertation, lead us to assume that $r_1 = 0.9$. On the other hand, since there are four letters in the DNA alphabet, $r_2 = \frac{1}{4}$ represent the average increment that can be obtain by adding a sequence to a partial score, due to the increment by adding a sequence is a binomial aleatory variable with parameters L y $p = \frac{1}{4}$. After some algebraic manipulations it is obtained

$$\frac{20}{l}\epsilon \leq 33i - 15t - 20K(i) + 2. \tag{6.7}$$

Now in order to have $\epsilon \geq 0$ is necessary that $C = 33i - 15t - 20K(i) + 2 \geq 0$. As a consequence, $\frac{i}{4} \leq K(i) \leq \frac{i}{4} + 1$. And therefore,

$$28i - 15t - 18 \leq C \leq 28i - 15t + 2. \tag{6.8}$$

The conclusion is that eliminations start around the level in which

$$i \geq \frac{15t + 18}{28} > \frac{t+1}{2} \tag{6.9}$$

of the search tree. This conclusion is consistent with the data in Table 6–12. As it can be seen, the first level in which the rule actually made eliminations is 4. This corresponds with the computation $i > \frac{5+1}{2} = 3$. In addition, by substituting $i = 4$ in to equation 6.7, the maximal value of ϵ turns out to be $\epsilon = 9.75$. This means that the interval of scores between 5 and 14.75 are the candidates to be bypassed by rule 6.4. If $i = 3$, the maximal value of ϵ is $\epsilon = 1.5$. The fact that this value

is very small, implies that minimal score is rarely founded. As a consequence, the experiment shows no candidates for bypassing in level 3.

6.5.3 A parallel DNA motif discoverer

Unfortunately, because of the depth-first search and job eliminations, BBM does not provide global information on the jobs and data loads that are unnecessary and those that remain to be explored, at any point in time. As a result, BBM does not render a natural parallelization. A way around this difficulty is replacing depth-first with breadth-first search and job eliminations. Such strategy leads to a dynamic programming algorithm (DPM). DPM traverses the levels of the same search tree as BBM. In order to do this, DPM represents the tree nodes of a level as a list. SCOW is used to segment the list of nodes into sublists which are distributed across the workers.

Single-master/worker algorithm

Upon reception of the sublist, the worker computes the scores of all the subtrees rooted at the sublist nodes; and uses a predicted *bestScore* and equation (6.4) to rule out the nodes that are worth keeping. Then returns the results to the master the best scores and the nodes that are worth further exploration. The master uses this information to predict a new *bestScore* and to create a new list of nodes, one that contains only nodes that have not been ruled out. Then, it runs SCOW to segment and distribute the new list across the workers. This process is repeated level by level in the search tree, until the last level is reached. This description encapsulates the Master/Worker implementation of DPM. Next is a high level description of the single-master/worker algorithm.

MASTER

- For each level 2 to t

  Use SCOW to distribute the job across the workers.

  Receive from the workers the sub-lists of vertices that have passed the rule.

Replace each parent in the sub-lists with their children.

Use the node with the best partial score to generate a new predictor.

- Return the alignment matrices with the best score.

WORKER

- For each level 2 to t

Receive the job from the master

Generate the sub-list of nodes that pass the rule

Send sublist to the master

Multiple-master/worker algorithm Although simpler to program, the Master/Worker method introduces some degree of indeterminism which cannot be absorbed by SCOW. Indeed, since no information on how many nodes will be ruled out from the list segments distributed to the workers is available by the time when SCOW was computed, the transmission times from the workers to the master, and the time spent by the master in the reception of the workers' information can only be estimated at that time. A solution to this problem is a Multi-master/Worker method, in which each worker becomes a master after finishing the processing its list segment. When the worker has ruled out all unnecessary nodes, it applies SCOW to the list segment of remaining nodes, and uses the result to distribute the job across the available workers. Thus, the Multi-master/Worker approach re-distributes the jobs in a multilevel tree-like fashion which is expected to reduce the makespan logarithmically.

STARTMASTER

Generate the list of the elements in level 2 of the tree.

Randomly select a branch in the tree and generate the first predictor.

Activate workers and Use SCOW to distribute the job across the workers.

While the job not finishes

Receive from Workers request for available workers and update the predictor

Update the available workers list and notify to the request worker

Receive from the workers the sub-lists

Send finish work order "deactivation" to all workers

Return the alignment matrices with the best score.

WORKER

While receive activation order

Receive job from activator

Perform the job

If level $<$ t

- Generate the data for next level.
- Request needed SCOW processor to the Start Master
- Activate workers and Use SCOW to distribute the job across the activate workers.

Else

- Send sublist to the Start Master

6.5.4 Preliminary results

In this section we summarize results of runs performed in a 64-node cluster. The measurements were conducted over a set of $t = 5$ DNA sequences, each of $n = 40$ characters. The motif's length, in turn, was set to $l = 5$.

Single master

Table 6–12 shows an average over three runs of the Master/Worker method.

Table 6–12: SCOW Single-Master

Parameter	Level 2	Level 3	Level 4	Level 5
Actual execution time	0.0101	0.0799	0.8282	4.3308
Time predicted by SCOW	0.0076	0.0697	0.7925	4.1138
Number of workers	2	3	9	16
Total number of tasks	36	1296	46656	362401
Cumulative percentage node remained	0%	0%	22%	2%

The difference between the execution times predicted by SCOW and the actual time are small since the unknown amount of information returned by the workers has been estimated statistically. The total execution time is the addition of the actual execution times, this is, 5.2390 seconds. The numbers of workers change because of the increments in the values of w_i, level by level. According with section 6.5.2, it is possible to generate the first $t/2$ levels in the search tree without using the bounding rule. This will save computing time. Although in this experiment there is just a marginal reduction of $0.09s$, in larger problems, the reduction can be much more significant.

Multi master

The performance of the Multi-master/Worker algorithm for the same problem was simulated numerically. The simulation yielded the statistics in Table 6–13.

Table 6–13: SCOW Multi-Master

Statistics	Level 2	Level 3	Level 4	Level 5
Masters	1	2	6	54
Workers per master	2	3	9	15(42),16(11), 18(1)
Longest compute time	0.0076	0.0402	0.1843	0.5068

The makespan of the Multi-master/Worker was 0.5547 seconds. The last row in this table shows an uneven distribution of workers per master. For instance, 42 out of the 54 masters have 15 workers, 11 have 16 workers, and only one master has 18 workers. This is a consequence of an uneven distributions of the node eliminations in the search tree. The logarithmic reduction in execution time is due to more accurate work and load distributions by SCOW, and the savings in communications and master processing of workers' information that occur in the single Master/Worker method.

CHAPTER 7
SOME ETHICAL ISSUES

The advent of information technology brings about a wide spectrum of freely available sources of scientific consultation. Several ethical issues arise in connection with this new information environment. First of all is the responsible use of the content in these information resources, especially in the recognition of authorship of the publications and the verification an validation of the presented results. The latter is especially important as some of these publications are not subjected to peer reviews and misrepresentations of scientific data have been the subject of legal prosecutions in the past. A remarkable case of such adulteration of scientific data is the one reported in [50] where a university professor, Dr. Poehlman of the university of Vermont, used false data to obtain scientific grants for total of 11.6 millions of dollars during a long period of time. According to the source [50] *"From in or about 1992 to 2000, Dr. Poehlman submitted seventeen (17) research grant applications to federal agencies or departments that included false and fabricated research data. In these grant applications, Dr. Poehlman requested approximately $11.6 million in federal research funding. In most cases, Dr. Poehlman falsified and fabricated research data in the "preliminary studies" sections of grant applications in order to support the scientific basis for and his expertise in conducting the proposed research. Reviewers of these grant applications relied on the accuracy of the "preliminary studies" to determine if a grant should be recommended for award. While many of the grant applications were not awarded, NIH and USDA expended approximately $2.9 million in research funding based on grant applications with false and fabricated*

research data". What is hard to estimate is the cascade effect of this false data in scientists who may have use it for their on research, as well as those whose grants where denied for lack of founding do to the misuse of federal grant resources. This potential damage cannot be repaired even with the strong sentence emitted in this case. Again, according to the source [50] this case was decided with the *"HHS actions against Dr. Poehlman include a life time debarment from receiving Public Health Service research funds and an agreement to retract or correct ten scientific articles due to research misconduct".*

Falsification or alteration of data can not only appear in big frauds like Dr. Poehlman's at also can be appreciated in more subtle situation where small, and therefore more difficult to detect, alterations of data can be used to sustain a false scientific claim. The consequences of such acts can be as damaging as the previously discussed case.

In this dissertation the principle of honest experiment design and data representation was fully observed. An example of this observance is the honest treatment of discrepancies between system measurements and initial assumptions made in the theoretical model that lead to the restriction of the model only to affine windows. Another example is the special care that was taken in performing the comparisons of all competing schedules on comparable grounds. This includes in particular, the design of a last round modification for schedules whose permanence was affected by the disparity of the workers end times, and the fact that comparisons were made with the same number of workers whenever the intrinsic characteristics of the scheduler allowed it. All referred works were duly cited and faithfully represented.

CHAPTER 8
CONCLUSIONS AND FUTURE WORKS

This dissertation studied the *load- and task-divisible jobs* (LTDJ) scheduling problem, divisible job referred to application that consist of a large number of independent core tasks, this core task can be agglomerate to produce data chunk of different sizes. This supposition makes the ability to model the problem as conventionally divisible load theory (DLT) but an additional discretization component is needed because the indivisibility of a single core tasks. In LTDJ application the communication time is an important factor in the scheduling, because this very large input. These applications represent a large number of important scientific computing problem, and achieve high performance when these applications are running using an efficient schedule. The research in this dissertation make significant contribution to the goal to achieving the maximal and efficiency system utilization. To this goal there are several challenges to solve

- Designee algorithms that taken in to account realistic model of the system.
- Designee algorithms that schedule the application with optimal throughput per period.
- Designee algorithms that schedule the application with efficient utilization of the computing capabilities of the network. This is, there will be no idle times between consecutive communications and computations and the resource use is optimal.
- Designee algorithms easy to use as part of a user-level scheduler for LTDJ applications.

In this dissertation several contributions are made to solve each of the above challenges.

Realistic model Unlike previous works whose basic models are focused in particular aspects of the targeted system, the basic mathematical model used in this dissertation takes into account all the operations that can possible occur in the execution of the master-worker SPMD program in a cluster. This mathematical model is also realistic in the sense that execution time of all the operations include both, startup and the execution times. Thus, execution times are presented as affine mappings on the number of agglomerated tasks. The constant part of the mapping corresponds to the operation's overhead. These mappings are obtained by regression over a sample of measurements of the system's behavior. Furthermore, the validity of these affine representations is taken in to account when identifying the schedule.

Maximal production Previous works have addressed the problem of obtaining a maximal throughput under fixed task sizes and using a linear mapping instead of an affine one for representing the time execution of the operations. In this dissertation the problem is cast in terms of affine mappings, and variable tasks sizes. In doing this, the concept of maximal throughput is replaced with that of maximal production, which is similar in spirit but more amenable for mathematical treatment.

Maximal Utilization This dissertation introduces a new metric for assessing the use of the cluster resources in the solution of a problem. The throughout revision in the literature that was performed for this dissertation showed that this metric has not been used before in the context of scheduling. Nonetheless, maximal utilization is an important attribute in the efficiency of a scheduler, specially in the cases where large cluster systems are available for an equally large number of users.

Resource selection Solving the resource selection problem is one of the most difficult steps in the design of a schedule. This dissertation presents two mathematically well-founded solutions to this problem. First, is a result that finds the best subset

of workers for achieving maximal production per period. Second, is the result that allows the identification of task sizes and number of workers that achieve the perfect worker utilization, in the sense that all the selected workers operate continuously throughout the execution of the job.

Maximal production and perfect utilization of the workers Another important contribution of this dissertation is the theoretical result that establishes that maximal productions and perfect workers utilization are achieved only in a finite number of values of the variable T representing the period. Furthermore, such values are shown to play a crucial role in the partition of the domain of the period. The intervals in this partition are characterized by the fact that the optimizations problems of maximal production and perfect utilization in a period can be formulated with a fix number of workers on each period. This number of workers is given by the underlying mathematics of the construction of the partition. This theoretical result is essential for the solvability of the make-span minimization problem. The strategy pursued in this dissertation consists in solving each of the finite optimization problems over the intervals in the partition, this is either the maximal production or perfect workers utilization in a period; and then searching for the solution that has minimal make-span.

Last round modification Although frequently mentioned as a measure of load balance, not all theoretical frameworks for scheduling whit maximal throughput per round include explicitly the condition that all the workers end operating at the same time. In this dissertation an explicit method for achieving this aim, called last round modification method, is presented both, in theory and implementation. Last round modification is proved to improve significantly the efficiency the schedule whenever the period is large.

Practical implementation Unlike many of the previous theoretical scheduling frameworks the scheduler identified with the theoretical frameworks presented in

this dissertation has been tested not only in simulation but also in actual problem solving situations. The result of real jobs confirm the theoretical and simulated superiority of the schedule built with the framework discussed in this dissertation.

Following is discuses four experimental presented in section 6.

8.1 Numerical comparison

This dissertation focused on the actual implementation of mathematical frameworks for master-worker schedulers. It this experiment is compared there such frameworks for multi-round schedulers, one for a periodic scheduler (SCOW) and one for a non-periodic scheduler (UMR). Two aspects were compared: the accuracy of the predictions computed with the mathematical framework, and the actual make-spans obtained in runs of a parallel motif finder. The experiments rendered a superior performance for tasks scheduled with the SCOW scheduler. The values predicted by the SCOW framework were also more accurate than those predicted by the UMR and Periodic framework.

8.2 An improved parallel brute force motif finding solver

It this part a parallel brute for algorithm to solve a bionformati motif finding problem is presented. In this experiment the running time and efficient of the platform use for two schedulers SCOW and FIFO is compared. Both aspects SCOW rendered the superior performance.

8.3 Experimental comparison with UMR

This experiment illustrates the limitations of a mathematical model frequently used to design master-worker schedulers. In fact, outside the interval of validity of the model, this is, the affine window, the parameters obtained by the frameworks did not represent the actual performance results. The best explanation for the lower performance and inaccuracy of predictions of the UMR framework is the fact that the chunk sizes obtained with this framework fall rapidly outside the affine window. From this stand point, the main advantage of a SCOW scheduler is its ability to

orchestrate the tasks on the basis of chunk sizes that remain within the affine window throughout the job's execution. A hybrid method was devised to take advantage of the good features of each scheduler. The hybrid method was illustrated with the motif finder algorithm, and proposed as a future work.

8.4 A Parallel Biosequence Motif Discoverer Based on Dynamic Programming

Several motif discovering algorithms as been proposed and developed, so far. As biodata increases in size and diversity, most of these methods have been pushed to the limit. Parallel computing appears as an alternative to cope with this increasingly higher demand for CPU time and memory. Unfortunately, most of the methods developed do not render a natural parallelization. Although still under development, the dynamic programming algorithm seems to offer a competitive parallel alternative to motif discovering. Dynamic programming is often feared because of its high memory demands. The Multi-master/Worker paradigm with SCOW may provide a way around this difficulty. SCOW is proved to be instrumental in balancing and overlapping communications and computations, while optimizing the use of memory space and CPU time.

8.5 Future works

There are many topic in scheduling area.

First idea is to incorporate this schedule in the construction of an application programming interface o software to execute LTDJ applications. This is necessary because the real user in several cases are interesting only in compute the job efficient but not in the algorithm or technique to construct a scheduler.

Develop mathematical model able to find the right balance between the use of UMR and SCOW in the hybrid method. The results shown in Table 6–11 were obtained with a good guess of this balance but not with the prove optimal.

Similarly, to develop a mathematical model able to find the right balance between the a initial phase with UMR, a second main phase with SCOW and the a last phase UMR2 (decreasing chuck sizes).

Due the superiority of Multi-master/Worker method section 6.5, develop a implementation of this method is needed, it is necessary incorporate a limit work distribution because the exponential resource demand

Finally, the theory develop has in to account heterogeneous resources. The experiment with this type of system are needed to complete the test for the SCOW schedule.

APPENDICES

APPENDIX A
ESPECIAL SCENARIOS

A.1 Solution to bus network

The *bus network* was describe in 3.1,that is, a StarAffine network such that all communication links have the same characteristics, the mapping $l_i = l$, similarly in the case when the $e + s$ dominate l asymptotically, them this resume in suppose $h_i = h$. Using the notation in this dissertation, the problem can be formulate as: "Given a time T, find $I \subset \{1, 2, ..., p\}$ such that

$$\text{Max } \{\sum_{i \in I} x_i \quad : \quad \bar{x} \text{ is } (T, p, F) - \text{partition of } X\} \tag{A.1}$$

$$\text{Subject to} \qquad \sum_{i \in I} h(x_i) \leq T \tag{A.2}$$

$$\max \{(r_i + w_i)(x_i) | i \in I\} \leq T\text{"} \tag{A.3}$$

In this case the problem is a special case of problem in section 4.1.3, in the theorem 14, is no characterize the case when two communication link are equal, because this theorem is only an approximation to the solution.

Suppose $h_i = h$ then $\sum_{i \in I} h(x_i) = h \sum_{i \in I} x_i + |I|H \leq T$, this expression show that if p is large then the best solution is achieve when the $|I|$ is minimum, that means, using a minimal set of workers.

If sort decreasing the maximal $x_i = \frac{T - W_i}{w_i}$ tasks processes by each workers, the solution is achieve by selecting the firsts q workers so that $\sum_{i=1}^{q} h(x_i) \leq T$ plus an additional workers if $T - \sum_{i=1}^{q} h_i(Y_i) > 0$.

Now the comparison of the maximal production for two workers is done, without loss of generality x_1 and x_2 is used. therefore

$$\frac{x_1 - x_2}{T} = \frac{1}{w_1} - \frac{1}{w_2} + \frac{\frac{W_2}{w_2} - \frac{W_1}{w_1}}{T} \tag{A.4}$$

them when T is arbitrary large the order in x_1 and x_2 become directly by the order in w_1 and w_2, that means, when T is large, $x_1 \geq x_2$, always that $w_1 \leq w_2$. furthermore "T large" can be characterize by $T \geq \frac{W_1 w_2 - W_2 w_1}{w_2 - w_1}$. The above discussion proof the next theorem.

Theorem 14. *Let p be a positive integer,*

1. *Sort the worker by increasing computation times. Renumber them so that $w_1 \leq w_2 \leq ... \leq w_p$.*
2. *Let $Y_i = (r_i + w_i)^{-1}(T)$ for $1 \leq i \leq p$ and q be the largest index so that $\sum_{i=1}^{q} h(Y_i) \leq T$. if $q < p$, let $T_\epsilon = T - \sum_{i=1}^{q} h_i(Y_i)$; otherwise, let $T_\epsilon = 0$*
3. *Then, if the period T is large enough the maximum value to* MaxTask(T) *for bus network is,*

$$MaxTask(T) = \sum_{i=1}^{q} Y_i + \max\{0, h_{q+1}^{-1}(T_\epsilon)\} \tag{A.5}$$

A.2 A good parallelization

For simplicity here only explain the criteria for the LinelAffine model.

Theorem 15. *With the above notation, Suppose that $h_i \geq w_i$, $1 \leq i \leq p$ and $h = min\{h_i | 1 \leq i \leq p\}$ then the solution to MP is reduce to 1 worker, and the maximal value for* MaxTask*(T) is*

$$\text{MaxTask}(T) = \frac{T}{h} \tag{A.6}$$

Proof Using the notation in this dissertation, the problem can be formulate as:

$$\text{Maximize} \qquad \textsc{MaxTask}(\mathrm{T}) = \sum_{i=1}^{p} x_i \tag{A.7}$$

$$\text{Subject to} \qquad \sum_{i=1}^{p} h_i.x_i \leq T \tag{A.8}$$

$$(r_i + w_i).x_i \leq T, \quad 1 \leq i \leq p \tag{A.9}$$

$$x_i \geq 0, \quad 1 \leq i \leq p \tag{A.10}$$

Let k =arg $min\{h_i | 1 \leq i \leq p\}$ and the $\bar{x}$ such that $x_k = \frac{T}{h_k}$ and $x_i = 0$ for $i \neq k$.

i)due to the initial condition $\frac{h_i}{w_i} \geq 1$, clearly $\bar{x}$ is solution of $\textsc{MaxTask}(\mathrm{T})$.

ii) In order to show that this solution $\bar{x}$ is maximal, let's consider the set of all optimal MP solutions. Let $\bar{x'}$ a tuple in this set. Them by equation A.8 is follow that,

$$\sum_{i=1}^{p} x'_i \leq \frac{\sum_{i=1}^{p} h_i.x'_i}{h_k} \leq \frac{T}{h_k} = x_k \tag{A.11}$$

Due to $\bar{x'}$ is an optimal solution then $\sum_{i=1}^{p} x'_i = x_k$, furthermore $\bar{x}$ is a optimal solution. □

APPENDIX B
OPERATING MODELS

There are a survey of model taken from [16]. We used through the dissertation the *full-overlap, single-port model.* but exit other model, we present the other model in this section. These models show below describe from the most powerful machines to the purely sequential processors.

Full overlap, multiple-port. In this first model, a processor can simultaneously receive data from all its neighbors, perform some (independent) computation, and send data to all of its neighbors. This model is not realistic if the number of neighbors is large.

Full overlap, single-port. In this second model, a processor node can simultaneously receive data from one neighbor, perform some (independent) computation, and send data to one neighbor. At any given time-step, there are at most two communications taking place, one incoming and one outgoing. This model is representative of a large class of modern machines and is the base model which we have already dealt with.

Receive-in-Parallel, single-port. In this third model, as in the next two, a processor node has one single level of parallelism: It can perform two actions simultaneously. In the this model, a processor can simultaneously receive data from one neighbor, and either perform some (independent) computation, or send data to one neighbor.

Send-in-Parallel, single-port. In this fourth model, a processor node can simultaneously send data to one neighbor and either perform some (independent) computation, or receive data from one neighbor.

Work-in-Parallel, single-port. In this fifth model, a processor node can simultaneously compute and execute a single communication, either sending to or receiving from one neighbor.

No internal parallelism. In this sixth and last model, a processor node can only do one thing at a time: either receiving from one neighbor, or computing, or sending data to one neighbor.

REFERENCE LIST

[1] The folding@home project. `http://folding.stanford.edu/`.

[2] Blast webpage. `http://http://www.ncbi.nlm.nih.gov/BLAST/`.

[3] Phu Luong, Clay P. Breshears, and Le N. Ly. Application of multiblock grid and dual-level parallelism in coastal ocean circulation modeling. *J. Sci. Comput.*, 20(2):257–277, 2004.

[4] John E. Dahlberg and Christian C. Mahler. The poehlman case: running away from the truth. *Science and Engineering Ethics*, 12(1):157–173, 2006.

[5] Kees Everaars and Barry Koren. Using coordination to parallelize sparse-grid methods for 3-d cfd problems. *Parallel Comput.*, 24(7):1081–1106, 1998.

[6] G. Shao, F. Berman, and R. Wolski. Master/slave computing on the grid. In *HCW '00: Proceedings of the 9th Heterogeneous Computing Workshop*, page 3, Washington, DC, USA, 2000. IEEE Computer Society.

[7] E. Heymann, M. Senar, E. Luque, and M. Livny. Adaptive scheduling for master-worker applications on the computational grid. In *GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pages 214–227, London, UK, 2000. Springer-Verlag.

[8] The top500 supercomputer list. `http://www.top500.org`.

[9] E. Coffman, editor. *Computer and Job-shop Scheduling Theory*. John Wiley & Sons Inc, 1976.

[10] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[11] Conway R., Maxwell W., and Miller L. *Theory of Scheduling*. Reading, MA: Addison-Wesley, 1967.

[12] Giorgio Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[13] B. Shirazi, K. Kavi, and A. Hurson, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[14] Pierre francois Dutot. Complexity of master-slave tasking on heterogeneous trees. *European Journal of Operational Research*, 164:2005, 2003.

[15] Wikipedia. `http://www.wikipedia.org/`.

[16] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Trans. Parallel Distrib. Syst.*, 15(4):319–330, 2004.

[17] O. Beaumont, A. Legrand, and Y. Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 98.2, Washington, DC, USA, 2003. IEEE Computer Society.

[18] Yang Yang and Henri Casanova. Umr: A multi-round algorithm for scheduling divisible workloads. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 24.2, Washington, DC, USA, 2003. IEEE Computer Society.

[19] T.L. Nguyen, S. Elnaffar, T. Katayama, and T.B Ho. `UMR2`: A better and more realistic scheduling algorithm for the grid. *international Conference on Parallel and Distributed Computing and Systems (PDCS'06)*, pages 432–43, 2006.

[20] Olivier Beaumont, Henri Casanova, Arnaud Legr, Robert Yves, and Yang Yang. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Trans. Parallel Distributed Systems*, 16, 2003.

[21] Yang Yang, Casanova Henri, Lawenda Marcin, and Legrand Arnaud. On the complexity of multi-round divisible load scheduling. Technical report, INRIA a CCSD electronic archive server based on P.A.O.L [http://hal.inria.fr/oai/oai.php] (France), 2007.

[22] Dimitris Bertsimas and David Gamarnik. Asymptotically optimal algorithms for job shop scheduling and packet routing, 1999.

[23] V. Bharadwaj, T. Robertazzi, and D. Ghose. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[24] R. Andonie, A. Chronopoulos, D. Grosu, and Galmeanu H. Distributed backpropagation neural networks on a pvm heterogeneous system. *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98)*, pages 555–560, 1998.

[25] O. Beaumont, A. Legrand, and Y. Robert. The master-slave paradigm with heterogeneous processors. *Transactions on Parallel and Distributed Systems*, 14(9):897–908, 2003.

[26] Olivier Beaumont, Arnaud Legr, Loris Marchal, and Yves Robert. Independent and divisible tasks scheduling on heterogeneous star-schaped platforms with limited memory. In *In PDP2005, 13th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 179–186. IEEE Computer Society Press, 2005.

[27] O. Beaumont, A. Legrand, Y. Robert, L. Carter, and J. Ferrante. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing*

Symposium, page 79, Washington, DC, USA, 2002. IEEE Computer Society.

[28] C. Lee and M. Hamdi. Parallel image processing applications on a network of workstations. *Parallel Comput.*, 21(1):137–160, 1995.

[29] D. Altilar and Y. Paker. An optimal scheduling algorithmfor parallel video processing. *In Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, 1998.

[30] V. Bharadwaj, D. Ghose, and V. Mani. Multi-installment load distribution in tree networks with delays. *IEEE Trans. Aerospace and Electronc Systems*, 1995.

[31] V. Bharadwaj, D. Ghose, and V. Mani. Multi-installment load distribution strategy for linear networks with communication delays. *Proceedings of the First International Workshop on Parallel Processing*, 1994.

[32] Cheng Y.-C. and Robertazzi T. Distributed computation with communication delay. *IEEE transactions on aerospace and electronic systems*, 1988.

[33] V. Bharadwaj, D. Ghose, and V. Mani. An efficient load distribution strategy for a distributed linear network of processors with communication delays. *Computer and Mathematics with Applications*, 1995.

[34] Cheng Y.-C. and Robertazzi T. Distributed computation for a tree-network with communication delay. *IEEE transactions on aerospace and electronic systems*, 1990.

[35] K. Li. Scheduling divisible tasks on heterogeneous linear arrays with applications to layered networks. *In Proceedings of the International Parallel and Distributed Processing Symposium*, 2002.

[36] M. Drozdowskir. Selected problems of scheduling tasks in multiprocessor computer systems. *University of Technology Press*, Series: Rozprawy, No. 321, 1997.

[37] J. Blazewicz and M. Drozdowski. Distributed processing of divisible jobs with communication startup costs. *Discrete Applied Mathematics*, 1997.

[38] M. Drozdowski and P. Wolniewicz. Divisible load scheduling in systems with limited memory. *Cluster Computing*, 6(1):19–29, 2003.

[39] Barbara Kreaseck. *Dynamic autonomous scheduling on heterogeneous systems.* PhD thesis, 2003. Co-Chair-Ferrante, Jeanne and Co-Chair-Carter, Larry.

[40] Y. Yang and H. Casanova. Extensions to the multi-installment algorithm: Affine costs and output data transfers. Technical report, July 2003.

[41] Y. Yang. *Scheduling divisible loads in multiple rounds.* PhD thesis, University of California at San Diego, La Jolla, CA, USA, 2005.

[42] J. Hein, S. Booth, and M. Bull. Exchanging multiple messages via mpi. Technical report, The HPCx Consortium, 2003.

[43] J. Blazewicz, M. Drozdowski, and M. Markiewicz. Divisible task scheduling - concept and verification. *Parallel Comput.*, 25(1):87–98, 1999.

[44] A. Rosenberg. Sharing partitionable workloads in heterogeneous nows: Greedier is not better. In *CLUSTER '01: Proceedings of the 3rd IEEE International Conference on Cluster Computing*, page 124, Washington, DC, USA, 2001. IEEE Computer Society.

[45] Y. Yang, K. van der Raadt, and H. Casanova. Multiround algorithms for scheduling divisible loads. *IEEE Trans. Parallel Distrib. Syst.*, 16(11):1092–1102, 2005.

[46] T.L. Nguyen, S. Elnaffar, T. Katayama, and T.B Ho. `MRRS`: A more efficient algorithm for scheduling divisible loads of grid applications. *IEEE/ACM International Conference on Signal-Image Technology and Internet-based Systems (SITIS'06)*, 2006.

[47] Optimization Software Lindo. `http://www.lindo.com`.

[48] D. Bertsekas, editor. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, Belmont, Mass., 1996.

[49] Institut national de recherche en informatique et automatique . `http://www.inria.fr/`.

[50] James Cheetham, Frank Dehne, Andrew Rau-Chaplin, Ulrike Stege, and Peter J. Taillon. Solving large fpt problems on coarse-grained parallel machines. *J. Comput. Syst. Sci.*, 67(4):691–706, 2003.

SCHEDULING DIVISIBLE TASKS UNDER PRODUCTION OR UTILIZATION CONSTRAINTS

Luis Fernando de la Torre Quintana
(787) 832-4040 EXT: 2638
Department of Electrical and Computer Engineering
Chair: Néstor Rodríguez
Degree: DOCTOR OF PHILOSOPHY
Graduation Date: July 2009

In this dissertation variants and extensions of ideas related to the scheduling of master-worker tasks on heterogeneous star networks are introduced. Some of these ideas were previously discussed in the form of theoretical frameworks for steady-state scheduling or as a divisible load theory. This dissertation combines some elements of these previous works to construct a new framework, and from it, an efficient algorithm (SCOW) for identifying a deterministic scheduler for clusters of workers. SCOW produces the parameters of a periodic user-level scheduler for a single-program multiple-data implementation of a master-worker parallel solution. SCOW minimizes the job make-span under either maximal production per period, or perfect worker utilization. The efficiency of the scheduler identified by SCOW is demonstrated through comparison with other schedulers, including those derived from the above mentioned theoretical frameworks. As shown in the simulation an actual computer runs, the scheduler identified by SCOW outperform in most cases those produced by the previous frameworks.