10

Scheduling of Divisible Loads on Heterogeneous Distributed Systems

Abhay Ghatpande[1]*, Hidenori Nakazato[1] and Olivier Beaumont[2]
[1]*GITI, Waseda University, Tokyo 169-0051*
[2]*LaBRI, France 33405*

1. Introduction

Divisible loads are a special class of applications that have regular linear structure, and which if given a large enough volume, can be partitioned into independently- and identically-processable *load fractions* (parts). Examples of applications that satisfy this divisibility property include image processing and rendering, signal processing, computation of Hough transforms, tree and database search, Monte Carlo simulations, computational fluid dynamics, and matrix computations.

The partitioning of a divisible load, the allocation (mapping) of the parts to appropriate processors for execution, and the sequencing (ordering) the transfer of the parts to and from the processors, is together known as *Divisible Load Scheduling* (DLS). *Divisible Load Theory* (DLT) is the framework that studies the optimization of DLS (Bharadwaj et al., 1996). Beaumont, Casanova, Legrand, Robert & Yang (2005) recently published a review of the work done to date in DLT. An exhaustive listing of papers regarding DLT and DLS is available on (Robertazzi, 2008).

1.1 Shortcomings of Traditional DLT

The basic principle of DLT to determine an optimal schedule for a master-slave system is the AFS (All slaves Finish Simultaneously) policy (Barlas, 1998). The AFS policy implies that after the nodes finish computing their individual load fractions, no results are returned to the source. This is an unrealistic assumption for many applications, as the result collection phase can contribute significantly to the total execution time. This is a serious shortcoming of traditional DLT. Along with the AFS policy, the presence of idle time in the optimal schedule has been overlooked in DLT work on result collection and heterogeneity. It is a very important issue because it may sometimes be possible to improve a schedule by inserting idle time.

A few papers that have dealt with DLS on heterogeneous systems to date (Beaumont, Marchal, Rehn & Robert, 2005; Beaumont et al., 2006; Beaumont, Marchal & Robert, 2005; Bharadwaj et al., 1996; Comino & Narasimhan, 2002; Rosenberg, 2001) proved that the sequence of allocation of data to the processors is important in heterogeneous networks. Without considering result collection, they proved that for optimum performance, (a) when processors have equal computation capacity, the optimal schedule results when the fractions are allocated in the order of decreasing communication link capacity, and (b) when communication capacity

*Corresponding author: abhay.ghatpande@ieee.org

is equal, the data should be allocated in the order of decreasing computation capacity. As far as can be judged, no paper has given a satisfactory solution to the scheduling problem where both the network bandwidth and computation capacities of the slaves are different, and the result transfer to the master is explicitly considered.

Cheng & Robertazzi (1990) and Bharadwaj et al. (1996, Chap. 3) addressed the issue of result collection with a simplistic constant result collection time, which is possible only for a limited number of applications on homogeneous networks. All other papers that have addressed result collection to date, advocated FIFO (First In, First Out) and LIFO (Last In, First Out) type of schedules. In FIFO, results are collected in the same order as that of load allocation, while in LIFO, the order of result collection is reversed. Barlas (1998) addressed the result collection phase for single-level and arbitrary tree networks, but the optimal sequences derived were essentially LIFO or FIFO. Rosenberg (2001) too proposed the LIFO and FIFO sequences for result collection. He concluded through simulations that FIFO is better when the network is homogeneous with a large number of processors, while LIFO is advantageous when the network is heterogeneous with a small number of processors.

For the first time, it was shown in (Beaumont, Marchal & Robert, 2005) that the LIFO and FIFO orderings are not always optimal for a given set of processors. In (Beaumont, Marchal, Rehn & Robert, 2005; Beaumont et al., 2006), it was proved that all processors from a given set of processors may not be used in the optimal solution. For the unidirectional single-port communication model (see Section 2), (Beaumont, Marchal, Rehn & Robert, 2005; Beaumont et al., 2006; Beaumont, Marchal & Robert, 2005) proved several interesting features in optimal schedules.

1.2 Chapter Organisation

Section 2 explains the choices made to represent the communication and computation speeds, the model used for size of result data, the assumptions and reasons regarding continuous delivery of data, the unidirectional one-port communication model, and the decision to use linear models of computation and communication time. Sections 2.3 and 3 provide a detailed derivation of the DLSRCHETS problem definition. After first laying the theoretical basis, the DLSRCHETS problem is defined in terms of a linear program. Section 4 lays the foundation of the two-slave system that forms the basis for the SPORT algorithm. Section 5 introduces the SPORT algorithm as a solution to the DLSRCHETS problem. Given a set of processors sorted in the order of decreasing communication speed, the complexity of SPORT is $O(m)$. Section 6 summarizes the chapter and ideas for future work.

2. The System Model

The execution of a divisible job on each slave comprises of three distinct phases in the following order — the allocation phase, where data is sent to the slave from the master, the computation phase, where the data is processed, and the result collection phase, where the slave sends the result data back to the source. The computation phase begins only after the entire load fraction allocated to that slave is received from the source. Similarly, the result collection phase begins only after the entire load fraction has been processed, and is ready for transmission back to the master. This is known as the *non-preemptive*, *atomic*, or *block based* model, and each phase forms a block on the time line as shown in Fig. 1.

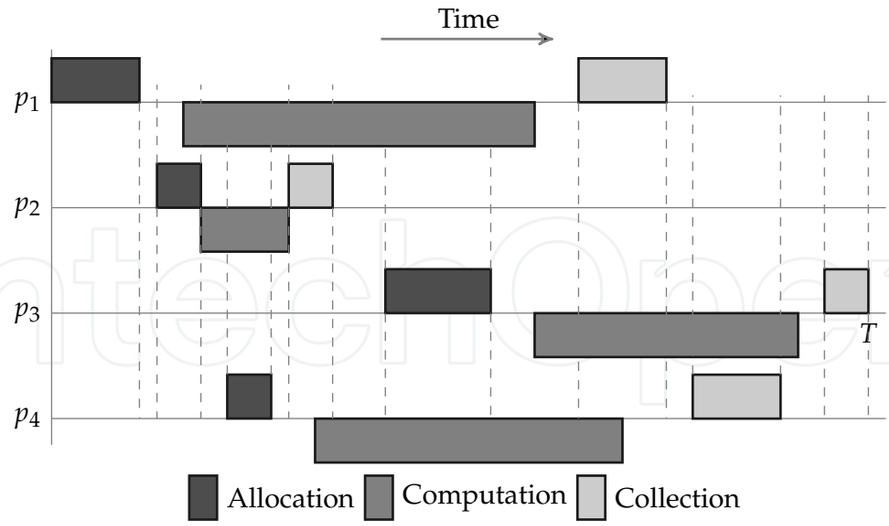

Fig. 1. A general schedule for DLSRCHETS. Processors can do only one thing at a time — either compute or communicate. There are three phases for each processor — allocation, computation, and result collection, in that order. However, phases of different processors may be interleaved. Each phase is *atomic*, i.e., continues to its end without interruption. Communication phases (either allocation or collection) cannot overlap as shown by the dashed lines. Computation phases are independent of each other.

2.1 Communication and Computation Model

The non-preemptive communication and computation phases necessitate that the slaves are continuously and exclusively available during the course of execution of the divisible load. The master and slaves can do only any one thing at a time — either communicate or compute (the no-overlap model), and if communicating, then either send data or receive data (the unidirectional one-port model).

A heterogeneous master-slave (sometimes called as *star* or *single-level tree*) system $\mathcal{H} = (\mathcal{P}, \mathcal{L})$ is as shown in Fig. 2, where $\mathcal{P} = \{p_0, \ldots, p_m\}$ is the set of $m + 1$ processors, and $\mathcal{L} = \{l_1, \ldots, l_m\}$ is the set of m network links that connect the master scheduler (source) p_0 at the center of the star (root of the tree), to the slave processors $p_1, \ldots, p_m$ at the points of the star (leaves of the tree). $\mathcal{E} = \{E_1, \ldots, E_m\}$ is the set of unit computation times of the slave processors, and $\mathcal{C} = \{C_1, \ldots, C_m\}$ is the set of unit communication times of the network links, i.e., p_k takes E_k time units to process a unit load transmitted to it from p_0 in C_k time units over the link l_k. It follows that $E_k, C_k > 0$, $k \in \{1, \ldots, m\}$. The values in $\mathcal{E}$ and $\mathcal{C}$ are assumed to be deterministic and available at the master.

The master holds a divisible load (job) $\mathcal{J}$ that is to be distributed and processed on $\mathcal{H}$. Based on the unit communication and computation time values of the slaves, the master p_0 splits $\mathcal{J}$ into parts (fractions) $\alpha_1, \ldots, \alpha_m$ and sends them to the respective slave processors $p_1, \ldots, p_m$ for computation. Each such set of m fractions is known as a *load distribution* $\alpha = \{\alpha_1, \ldots, \alpha_m\}$. The source does not retain any part of the load for computation. Since the job $\mathcal{J}$ is assumed to be arbitrarily divisible, $\alpha_k \in \mathbb{R}_0^+$, $\alpha_k \geq 0$, $k \in \{1, \ldots, m\}$. The unit communication and computation times are conditional upon the job $\mathcal{J}$ under consideration. So ideally, the values should be indexed as $C_k^{\mathcal{J}}$ and $E_k^{\mathcal{J}}$, to indicate that the values are valid only for the job $\mathcal{J}$. This index is omitted as the context is clear to be the job $\mathcal{J}$.

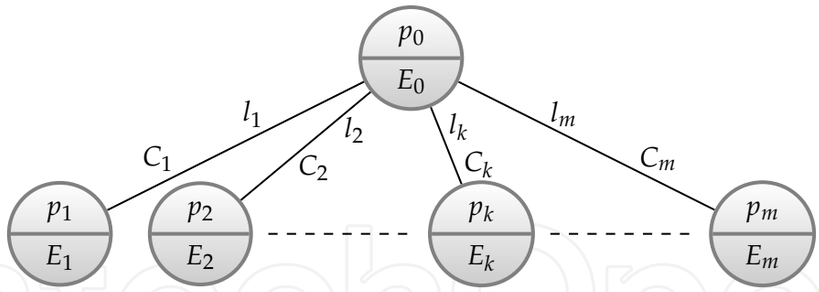

Fig. 2. The heterogeneous master-slave system $\mathcal{H}$. The processors have different computation speeds and network bandwidths.

2.2 Result Data Model

For the divisible loads under consideration, the computation phase usually involves simple linear transformations on the input data, and the volume of returned results can be considered to be proportional to the amount of load received in the allocation phase. If the allocated load fraction is α_k, then the returned result is equal to $\delta\alpha_k$, $0 \leq \delta \leq 1$. The constant δ is application specific, and is the same for all processors for a particular load $\mathcal{J}$. This is the accepted model for returned results in literature to date (Adler et al., 2003; Barlas, 1998; Beaumont, Marchal, Rehn & Robert, 2005; Beaumont et al., 2006; Beaumont, Marchal & Robert, 2005; Bharadwaj et al., 1996; Comino & Narasimhan, 2002; Rosenberg, 2001; Yu & Robertazzi, 2003).

2.3 Communication and Computation Time

The time taken for communication and computation is assumed to be a linearly increasing function of the size of load fraction. For a load fraction α_k, $\alpha_k C_k$ is the transmission time from p_0 to p_k, $\alpha_k E_k$ is the time it takes p_k to perform the requisite processing on α_k, and $\delta\alpha_k C_k$ is the time it takes p_k to finally transmit the results back to p_0. Though a linear model is considered for computation and communication times for the sake of simplicity, all results can be easily extended to other models.

In the DLSRCHETS problem, the master has to partition the load $\mathcal{J}$ into fractions $\alpha_1, \ldots, \alpha_m$, and manage the allocation of these fractions to, and collection of the results from the processors $p_1, \ldots, p_m$ in the minimum possible time. Let $\mathcal{T} = \{1, \ldots, m\}$ be the set of *tasks* corresponding to the m fractions that are allocated to, and $\mathcal{R} = \{1, \ldots, m\}$ be the set of *results* that are collected from the processors $p_1, \ldots, p_m$ respectively.

Though the load fractions (tasks) can be processed independently of each other on the respective processors, the single-port communication model implicitly induces a *precedence order* on the distribution of the tasks and collection of the results. Let $\prec_a$ and $\prec_c$ be *total orders* on the sets $\mathcal{T}$ and $\mathcal{R}$ respectively, such that $\prec_a$ represents the sequence (order) in which processors are allocated tasks, and $\prec_c$ is the sequence in which results are collected from the processors at the master. Then, $i \prec_a j$ implies that task i *precedes* task j (or equivalently task j *succeeds* task i) in the allocation sequence $\prec_a$, and $i \prec_c j$ signifies that result i precedes result j in the collection sequence $\prec_c$. If $\{k \in \mathcal{T} : i \prec_a k \prec_a j\} = \emptyset$, then task i is the *immediate predecessor* of task j in $\prec_a$, and is denoted as $i \preccurlyeq_a j$. Similarly, if $\{k \in \mathcal{R} : i \prec_c k \prec_c j\} = \emptyset$, then result j is the *immediate successor* of result i in $\prec_c$, and is denoted as $i \preccurlyeq_c j$. Define $B^i_{\prec_a} := \{j \in \mathcal{T} : j \prec_a i\} \cup \{i\}$ and $F^i_{\prec_a} := \{j \in \mathcal{T} : i \prec_a j\} \cup \{i\}$, i.e., $B^i_{\prec_a}$ is the set of task i and the tasks *before* i (*predecessors* of i) in $\prec_a$, while $F^i_{\prec_a}$ is the set of task i and the *followers* (*successors*) of task i in $\prec_a$. $B^i_{\prec_c}$ and $F^i_{\prec_c}$ are defined accordingly for $\prec_c$. The *minimal element* of $\prec_a$ is defined as $\prec_a^+ := \exists!\, i \in \mathcal{T} : B^i_{\prec_a} = \{i\}$ and the *maximal element* of $\prec_a$ is defined as, $\prec_a^- := \exists!\, i \in \mathcal{T} : F^i_{\prec_a} = \{i\}$, i.e., $\prec_a^+$ and $\prec_a^-$ are

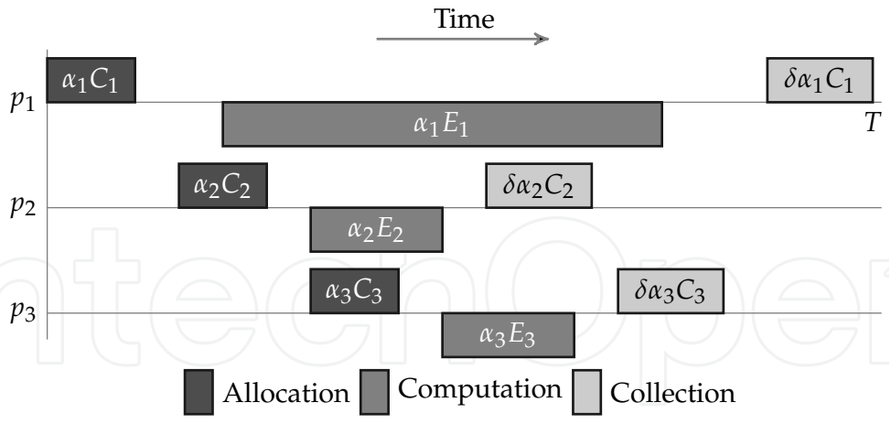

Fig. 3. A possible schedule with $m = 3$. The three phases of each processor are atomic and satisfy the constraints (1) to (9).

the first and last tasks allocated in $\prec_a$. $\prec_c^+$ and $\prec_c^-$ are similarly defined as the first and last results returned in $\prec_c$.

For a given load $\mathcal{J}$, the objective is to minimize the total processing time T, which is defined as the time taken from the point when the master first initiates the allocation of tasks, to the point when the master completes reception of all the results. The *schedule* $\mathcal{S}$ of DLSRCHETS for a given load distribution α, is a pair (t, r), where, $t : \mathcal{T} \mapsto \mathbb{R}_0^+$ is the task allocation start time function, and $r : \mathcal{R} \mapsto \mathbb{R}_0^+$ is the result collection start time function. In a *feasible* schedule, the start times in t and r must satisfy the following constraints:

$$t_j - t_i \geq \alpha_i C_i \qquad \forall\, i \in \{1, \ldots, m\}, i \preccurlyeq_a j \tag{1}$$

$$t_i \geq \sum_{j \in B^i_{\prec_a} \setminus \{i\}} \alpha_j C_j \qquad \forall\, i \in \{1, \ldots, m\} \tag{2}$$

$$r_j - r_i \geq \delta\alpha_i C_i \qquad \forall\, i \in \{1, \ldots, m\}, i \preccurlyeq_c j \tag{3}$$

$$T - r_i \geq \sum_{j \in F^i_{\prec_c}} \delta\alpha_j C_j \qquad \forall\, i \in \{1, \ldots, m\} \tag{4}$$

$$r_i - t_i \geq \alpha_i C_i + \alpha_i E_i \qquad \forall\, i \in \{1, \ldots, m\} \tag{5}$$

$$t_i \neq r_j \qquad \forall\, i, j \in \{1, \ldots, m\} \tag{6}$$

$$r_j - t_i \geq \alpha_i C_i \qquad \forall\, j \in \{1, \ldots, m\}, \forall\, t_i < r_j \tag{7}$$

$$t_i - r_j \geq \delta\alpha_j C_j \qquad \forall\, i \in \{1, \ldots, m\}, \forall\, r_j < t_i \tag{8}$$

$$t_i, r_j \geq 0 \qquad \forall\, i, j \in \{1, \ldots, m\} \tag{9}$$

The precedence constraints of $\prec_a$ are enforced by (1) and (2), while inequalities (3) and (4) impose the precedence constraints of $\prec_c$ and define the processing time T. The fact that the result collection cannot begin before the execution of the entire load fraction is complete is shown by (5). Constraints (6), (7), and (8) impose the single-port model so that no allocation and collection phase can overlap. The non-negativity of the start times is ensured by (9).

Figure 3 shows the timing diagram for a feasible schedule with $m = 3$. The time spent in communication with the master p_0 is shown above the horizontal axes, and time spent in computation by the individual processors below the horizontal axes. Since p_0 does not retain any part of the load for itself, there is no p_0 axis.

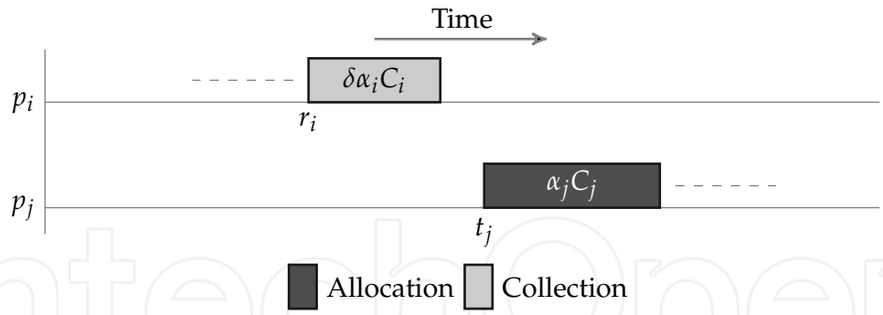

Fig. 4. Interleaved result collection. There exists at least one pair of r_i and t_j that immediately follow each other.

Condition 1 (Allocation Precedence Condition)**.** The master should first allocate the entire load to the processors before receiving any results from the processors.

Lemma 1 (Allocation Precedence Lemma)**.** *There exists an optimal schedule for DLSRCHETS that satisfies the allocation precedence condition. (There may exist other optimal schedules that do not satisfy the allocation precedence condition.)*

Proof. Consider a feasible schedule with processing time T, that satisfies (1) to (9) for a load distribution α, and an arbitrary order of allocation and collection $\prec_a$ and $\prec_c$, such that some results are collected before the load is completely allocated first.
Then, there exists at least one pair (i, j) with $i \prec_a j$, such that the result collection starting at r_i is followed by a task allocation at t_j, without any other intermediate communication phase as shown in Fig. 4.
Suppose that all load fractions in α, and all other start times in t and r are maintained the same, and only the order of collection of result i and allocation of task j is exchanged, such that the new allocation start time of task j is $t'_j = r_i$, and the new collection start time of result i is $r'_i = r_i + \alpha_j C_j$.
Since the above exchange does not alter the order of allocation of different tasks, the precedence constraints of $\prec_a$ defined by (1) and (2) still hold. Similarly, the precedence constraints of $\prec_c$, imposed by (3) and (4) also hold after the exchange. The constraints (6), (7), and (8) are valid after the exchange because the single-port model is not violated by the exchange.
Only the conditions expressed by (5) require verification. Before the exchange, the conditions $r_i - t_i \geq \alpha_i C_i + \alpha_i E_i$ and $r_j - t_j \geq \alpha_j C_j + \alpha_j E_j$ are satisfied. After the exchange, the constraints (5) are still valid because $r'_i - t_i = r_i + \alpha_j C_j - t_i > r_i - t_i$, and $r_j - t'_j = r_j - r_i > r_j - t_j$.
From the above observations, it is clear that after the reordering, all conditions for feasibility are still satisfied. Moreover, the orders $\prec_a$ and $\prec_c$ are unchanged, and no additional processing time is required for the reordering.
If a similar reordering is carried out for all such pairs (i, j), then the allocation precedence condition is satisfied with no addition in total processing time T.
Now if there is an optimal schedule for DLSRCHETS that does not satisfy the allocation precedence condition, then a reordering can be performed as mentioned above so that the schedule satisfies the allocation precedence condition without an increase in the total processing time. That is, there always exists an optimal schedule that satisfies the allocation precedence condition, and only such schedules need be considered in the search for the optimal schedule. ■

Two other basic lemma are stated before the DLSRCHETS problem is defined.

Lemma 2. *There exists an optimal schedule for DLSRCHETS that has no idle time between any two consecutive allocation phases and any two consecutive result collection phases. (There may exist other optimal schedules that do not satisfy this condition.)*

Proof. Assume that a feasible schedule that obeys (1) to (9), and in addition also satisfies the allocation precedence condition, has idle time between the consecutive communication phases (see Fig. 3). Let the processing time be T, the load distribution be α, and $(\prec_a, \prec_c)$ be the orders of allocation and collection.
According to the assumptions in the system model, all processors are available continuously and exclusively during the entire execution process, and the master can only communicate with one processor at a time. For any $i \preccurlyeq_a j$, when processor p_i completes the reception of its allocated task at time $t_i + \alpha_i C_i$, processor p_j is already available and can start receiving data immediately at $t_j = t_i + \alpha_i C_i$. Because the schedule satisfies the allocation precedence condition, load is first distributed to all the processors sequentially before result collection begins. Thus the start time of each task $i \in \mathcal{T}$ can be brought forward so that $t_i = t_{\prec_a^+} + \sum_{j \in B^i_{\prec_a} \setminus \{i\}} \alpha_j C_j$, and the inequalities (1) and (2) are reduced to equalities without exceeding T.
Following a similar logic to the one above, the result collection of each result $i \in \mathcal{R}$ can be delayed to the extent necessary to make the result collection start time $r_i = T - \sum_{j \in F^i_{\prec_c}} \delta \alpha_j C_j$, with inequalities (3) and (4) reduced to equalities and no extra time added to T.
Since any feasible schedule can be reordered in this manner to eliminate the idle time between communication phases, it follows that an optimal schedule to DLSRCHETS also has no idle time between any two consecutive allocation and result collection phases. ■

Lemma 3. *There exists an optimal schedule for DLSRCHETS that has no idle time between the allocation and computation phases of each processor. (There may exist other optimal schedules that do not satisfy this condition.)*

Proof. Following an argument similar to the one used in Lemma 2, since all processors are always available, they can begin computing immediately upon receiving their load fractions in the allocation phase without affecting the schedule.
Any processor p_i begins computing its allocated task at time $t_{\prec_a^+} + \sum_{j \in B^i_{\prec_a}} \alpha_j C_j$ without crossing the time interval T. Since any feasible schedule can be reordered in this manner, an optimal schedule to DLSRCHETS too has no idle time between the allocation and computation phases of each processor. ■

Theorem 1 (Feasible Schedule Theorem)**.** *There exists an optimal schedule for DLSRCHETS that satisfies Lemmas 1 to 3.*

Proof. If there exists an optimal schedule that does not satisfy any or all of the Lemmas 1 to 3, it can always be reordered as explained in the respective proofs to satisfy the same. ■

From Theorem 1, it follows that only those schedules that satisfy Lemmas 1 to 3 need be considered in the search for the optimal solution to DLSRCHETS. A possible timing diagram for such a schedule is shown in Fig. 5.
From the preceding discussion, it can be concluded that the start times t and r in the optimal schedule for DLSRCHETS can be determined from the sequences $\prec_a$ and $\prec_c$, and the load distribution α that minimize the processing time T. Hence instead of finding t and r as in traditional scheduling practice, the DLSRCHETS problem is formulated as a linear programming

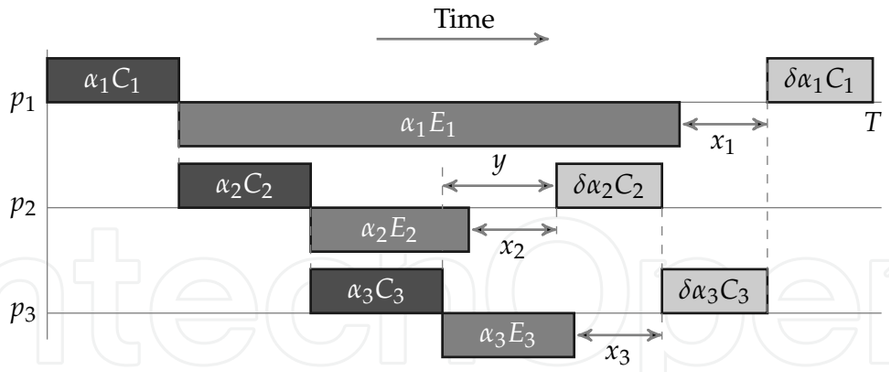

Fig. 5. A schedule for $m = 3$ that satisfies the Feasible Schedule Theorem. Result collection begins only after the entire load is distributed. Each allocation and result collection phase follows its predecessor without delay. The computation phase of each processor follows its allocation phase without delay. Idle time may be present in each processor between the end of its computation phase and the start of the result collection phase.

problem, to find $\prec_a$, $\prec_c$, and α that minimize T. Once the optimal values of these variables are known, it is straightforward to find the optimal schedule.
The constraints (1) to (9) and the allocation precedence condition are combined into a unified form, and for each processor p_i, constraints on T are written in terms of $B^i_{\prec_a}$ and $F^i_{\prec_c}$. The DLSRCHETS problem is defined in terms of a linear program as follows.

Definition 1 (Divisible Load Scheduling with Result Collection on HETerogeneous Systems)**.**

Given a heterogeneous network $\mathcal{H} = (\mathcal{P}, \mathcal{L})$, a divisible load $\mathcal{J}$, unit communication and computation times $\mathcal{C}$, $\mathcal{E}$, find the sequence pair $(\prec^*_a, \prec^*_c)$, and load distribution $\alpha^* = \{\alpha^*_1, \ldots, \alpha^*_m\}$ that

Minimize T
Subject To:

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j + \alpha_k E_k + \sum_{j \in F^k_{\prec_c}} \delta\alpha_j C_j \leq T \qquad k = 1, \ldots, m \tag{10}$$

$$\sum_{j=1}^{m} \alpha_j C_j + \sum_{j=1}^{m} \delta\alpha_j C_j \leq T \tag{11}$$

$$\sum_{j=1}^{m} \alpha_j = \mathcal{J} \tag{12}$$

$$T \geq 0, \quad \alpha_k \geq 0 \qquad k = 1, \ldots, m \tag{13}$$

In the above formulation, for a sequence pair $(\prec_a, \prec_c)$, and a load distribution α, the LHS (Left Hand Side) of constraint (10) indicates the total time spent in transmission of tasks to all the processors that must receive load before the processor p_i can begin processing its allocated task, the computation time on the processor p_i itself, and the time for transmission back to the master of results of processor p_i, and all its subsequent result transfers. For the no-overlap model to be satisfied, the processing time T should be greater than or equal to this time for all the m processors. The single-port communication model is enforced by (11)

since its LHS represents the lower bound on the time for distribution and collection under this model. The fact that the entire load is distributed amongst the processors is imposed by (12). This is the *normalization equation*. The non-negativity of the decision variables is ensured by constraint (13).

3. Analysis of Optimal Solution

Processors that are allocated load are called *participating processors* or *participants*.

Theorem 2 (Idle Time Theorem)**.** *There exists an optimal solution to the DLSRCHETS problem, in which irrespective of whether load is allocated to all available processors, at the most one of the participating processors has idle time, and the idle time exists only when the result collection begins immediately after the completion of load distribution.*

Proof. For a pair $(\prec_a, \prec_c)$, the DLSRCHETS problem defined by (10) to (13) always has a feasible solution. This is because, for any load distribution α that satisfies (12), T can be made arbitrarily large to satisfy the inequalities (10) and (11). It implies that the polyhedron formed by the constraints of the DLSRCHETS problem, $P := \{\boldsymbol{x} \in \mathbb{R}^{m+1} : A\boldsymbol{x} \leq \boldsymbol{b},\ \boldsymbol{x} \geq 0\} \neq \emptyset$.
According to the theory of linear programming, the optimal solution to DLSRCHETS is obtained at some vertex of this polyhedron (Dantzig, 1963; Vanderbei, 2001). As the DLSRCHETS problem has $m+1$ decision variables and $2m+3$ constraints, in a *non-degenerate* optimal solution, at the optimal vertex, $m+1$ constraints out of these must be *tight*, i.e., satisfied with equality. In a *degenerate* optimal solution, more than $m+1$ constraints are tight.
It is clear that in an optimal solution, the normalization constraint (12) will always be tight, and T will always be greater than zero. This means that m constraints out of the remaining $2m+1$ constraints will be tight in a non-degenerate optimal solution. There are two possible ways to proceed with the analysis at this point depending on the allocated load fractions in the optimal solution.

1. $\forall\, \boldsymbol{k} \in \{\boldsymbol{1}, \ldots, \boldsymbol{m}\} : \boldsymbol{\alpha_k} > \boldsymbol{0}$.
   In this case, all the load fractions are assumed to be always greater than zero, i.e. number of participants is m. Since all decision variables are positive, there can be no degeneracy (Vanderbei, 2001, Chapter 3).
   It leaves only $m+1$ constraints (10) and (11), out of which m will be tight in the optimal solution. Hence, in the optimal solution, either,
   (a) the m constraints (10) are tight, and the (11) constraint is not, or
   (b) the (11) constraint is tight and one of the (10) constraints is not.

   If any constraint from (10) and (11) is not tight in the optimal solution, it implies a *shortfall* in the LHS as compared to the optimal processing time. In constraints (10) this shortfall represents idle time in a processor, while in (11) it represents the intervening time interval between completion of load distribution from the master and the start of result transfer to the master.
   Thus, if the option (a) above is true, then none of the processors have any idle time in the optimal solution. If the option (b) is true, then one of the processors has idle time, and since this happens only when constraint (11) is tight, it means that idle time in a processor exists only when result transfer to the master begins immediately after completion of load allocation is completed. This is similar to the analysis in Beaumont, Marchal, Rehn & Robert (2005); Beaumont et al. (2006).

2. $\exists\, k \in \{1, \ldots, m\} : \alpha_k = 0$.
In this case, some of the processors can be allocated zero load in the optimal solution. The analysis has two parts — one for non-degenerate and the other for degenerate optimal solutions.
Non-degenerate Optimal Solution
If there are p $(p \leq m)$ participants in the optimal solution,then $m - p$ constraints of (13) are necessarily tight. This means that out of the $m + 1$ constraints (10) and (11), only p constraints will be tight in the optimal solution. Hence, in an optimal solution, either,
   (a) p of the (10) constraints are tight, $m - p$ of the (10) constraints are not tight, and the (11) constraint is not tight, or
   (b) the (11) constraint is tight, $p - 1$ of the (10) constraints are tight, and $m - p + 1$ of the (10) constraints are not tight.

   In the optimal solution, if the option (a) is true, then $m - p$ processors have idle time, while if the option (b) is true, then $m - p + 1$ processors have idle time.
   Since $m - p$ processors are not allocated load, it is obvious that they are idle throughout in either of the above two options. The additional processor with idle time if the option (b) is true has to be one of the participating processors. This means that idle time in a participating processor exists only when the result collection begins immediately upon completion of load allocation.
   Degenerate Optimal Solution
   Similar to the non-degenerate case, if there are p $(p \leq m)$ participants in the optimal solution, then $m - p$ constraints of (13) are necessarily tight. Since the optimal solution is degenerate, *more than* p constraints out of the $m + 1$ constraints (10) and (11) will be tight.
   This means that in the optimal solution, irrespective of whether the (11) constraint is tight, *at least* p of the (10) constraints are tight, and *less than* $m - p$ of the (10) constraints are not tight. Since $m - p$ processors are necessarily idle, some of the (10) constraints corresponding to the processors allocated zero load are tight in the degenerate solution. Since $\forall\, k \in \{1, \ldots, m\},\ B^k_{\prec_a}, F^k_{\prec_c} \subseteq \{1, \ldots, m\}$, it implies that,

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j \leq \sum_{j=1}^{m} \alpha_j C_j \qquad k \in \{1, \ldots, m\}$$

and

$$\sum_{j \in F^k_{\prec_c}} \delta\alpha_j C_j \leq \sum_{j=1}^{m} \delta\alpha_j C_j \qquad k \in \{1, \ldots, m\}$$

It follows that,

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j + \sum_{j \in F^k_{\prec_c}} \delta\alpha_j C_j \leq \sum_{j=1}^{m} \alpha_j C_j + \sum_{j=1}^{m} \delta\alpha_j C_j \qquad k \in \{1, \ldots, m\} \tag{14}$$

If (11) is not tight, then the RHS (Right Hand Side) of (14) is strictly less than T. That is,

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j + \sum_{j \in F^k_{\prec_c}} \delta\alpha_j C_j < T \quad k \in \{1, \ldots, m\} \tag{15}$$

If $\exists\, k \in \{1,\ldots,m\} : \alpha_k = 0$, then $\alpha_k E_k = 0$, and from (15), it immediately follows that the corresponding constraint from (10) can never be tight.
Thus, a constraint corresponding to a processor p_k allocated zero load is tight in the optimal solution only if

$$\sum_{j \in B^k_{\prec_a}} \alpha_j C_j + \sum_{j \in F^k_{\prec_c}} \delta \alpha_j C_j - T = 0 \tag{16}$$

or equivalently if (14) is satisfied with an equality, *and* the RHS of (14) is equal to T, i.e, the (11) constraint is tight.
It is now clear that a degenerate optimal solution exists only when the (11) constraint is tight, and the condition (16) is satisfied. To find when the condition is satisfied, consider the case where for some pair $(\prec_a, \prec_c)$, one or more of the processors allocated zero load follow each other at the end of the allocation sequence and the start of the result collection sequence in the optimal solution.
For example, if $\alpha_i, \alpha_j, \alpha_k = 0$, and one or more of the following occur (the list is not exhaustive):

- $\prec_a^- = i$ and $\prec_c^+ = i$
- $i \preccurlyeq_a j$, $\prec_a^- = j$ and $\prec_c^+ = i$
- $i \preccurlyeq_a j$, $\prec_a^- = j$, $\prec_c^+ = k$ and $k \preccurlyeq_c i$

Only if such tail-end zero-load processors exist, then (14) is satisfied with an equality. Finally, if constraint (11) is tight in the optimal solution, then it follows that the constraints corresponding to these processors are tight.
The linear program obtained after eliminating the redundant constraints corresponding to the tail-end zero-load processors has a non-degenerate optimal solution. This is because, the feasible region defined by the constraints of the non-degenerate problem does not change after addition of the redundant constraints. Hence only a single participant processor has idle time in the degenerate optimal solution.

From the preceding discussion on the optimal solution to the linear program for a pair $(\prec_a, \prec_c)$, it follows that in the optimal solution to the DLSRCHETS problem, $(\prec_a^*, \prec_c^*, \alpha^*)$, at the most one participating processor can have idle time. The idle time occurs *only when* the result collection from processor $\prec_c^+$ starts immediately after completion of load allocation to processor $\prec_a^-$. ■

There are $m!$ possible permutations each of $\prec_a$ and $\prec_c$, and the linear program has to be evaluated $(m!)^2$ times to determine the globally optimum solution $(\prec_a^*, \prec_c^*, \alpha^*)$ for DLSRCHETS. Since the solution to the linear program is completely determined by the values of δ, $\mathcal{C}$ and $\mathcal{E}$, along with the pair $(\prec_a, \prec_c)$, it is not possible to predict which of the processors or how many processors will be allocated zero load.

4. Analysis of Two-Slave System

For a sequence pair (σ_a, σ_c) and load distribution $\alpha = \{\alpha_1, \ldots, \alpha_m\}$, a slave processor p_i, may have idle time x_i because it may have to wait for another processor to release the communication medium for result transfer (ref. Fig. 5). In the optimal solution to DLSRCHETS, $\forall i \in \{1 \ldots m\}$, $x_i = 0$, if and only if $y > 0$, and that there exists a unique $x_i > 0$ if and only if $y = 0$, where y is the intervening time interval between the end of allocation phase of processor $\sigma_a[m]$ and the start of result collection from processor $\sigma_c[1]$. For the FIFO schedule in

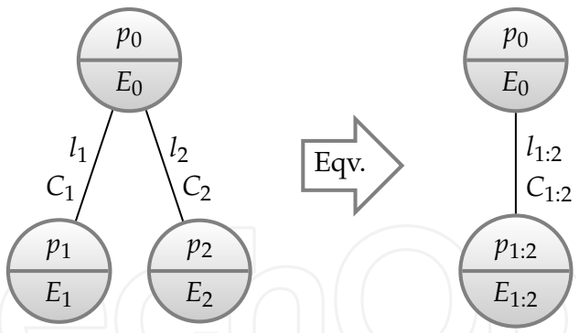

Fig. 6. The heterogeneous two-slave system. The two processors p_1 and p_2 are replaced by an equivalent virtual processor $p_{1:2}$ on the right. The two network links l_1 and l_2 are replaced by an equivalent virtual link $l_{1:2}$. As far as the master p_0 is concerned, there is no difference in the time it takes for the equivalent processor to execute a task.

particular, processor $\sigma_a[m]$ can always be selected to have idle time when $y = 0$, i.e., in the FIFO schedule, $x_{\sigma_a[m]} > 0$ if and only if $y = 0$. In the LIFO schedule, since $y > 0$ always, no processor has idle time, i.e., $\forall i \in \{1 \dots m\}$, $x_i = 0$ always (Beaumont, Marchal, Rehn & Robert, 2005; Beaumont et al., 2006; Beaumont, Marchal & Robert, 2005).
Let the allocation sequence be represented by σ_a, and the collection sequence by σ_c, both of which are permutations of the index set $K = \{1, \dots, m\}$ of slave processors in the heterogeneous system $\mathcal{H}$. For a pair (σ_a, σ_c), the solution to the linear program defined by (10) to (13) is completely determined by the values of δ, $\mathcal{E}$, $\mathcal{C}$, and it is not possible to predict which processor is the one that has idle time in the optimal solution. In fact, it is possible that not all processors are allocated load in the optimal solution, in which case some processors are idle throughout.
The heterogeneous system $\mathcal{H} = (\mathcal{P}, \mathcal{L})$ with $m = 2$ is shown in Fig. 6. It is defined by $\mathcal{P} = \{p_0, p_1, p_2\}$ and $\mathcal{L} = \{l_1, l_2\}$. The unit computation and communication times are defined by the sets $\mathcal{E} = \{E_1, E_2\}$, and $\mathcal{C} = \{C_1, C_2\}$. Without loss of generality, it is assumed that the total load to be processed available at the master is $\mathcal{J} = 1$. Also it is assumed that $C_1 \leq C_2$. No assumptions are possible regarding the relationship between E_1 and E_2, or $C_1 + E_1 + \delta C_1$ and $C_2 + E_2 + \delta C_2$.
An important parameter, ρ_k, known as the *network parameter* is introduced, which indicates for a slave p_k, how fast (or slow) its computation parameter E_k is with respect to the communication parameter C_k of its network link:

$$\rho_k = \frac{E_k}{C_k} \qquad k = 1, \dots, m \tag{17}$$

The master p_0 distributes the load $\mathcal{J}$ between the two slave processors p_1 and p_2 so as to minimize the processing time T. Depending on the values of δ, $\mathcal{E}$ and $\mathcal{C}$, there are three possibilities:

1. **Entire load is distributed to p_1 only.**
   The total processing time is given by

$$T^1 = C_1 + E_1 + \delta C_1 = C_1(1 + \delta + \rho_1) \tag{18}$$

2. **Entire load is distributed to p_2 only.**

The total processing time in this case is

$$T^2 = C_2 + E_2 + \delta C_2 = C_2(1 + \delta + \rho_2) \tag{19}$$

3. **Load is distributed to both p_1 and p_2.**
It can be proved that as long as $C_1 \leq C_2$, only the schedules in Figs. 7, 8, and 9 can be optimal for a two-slave system. These schedules are the FIFO schedule, the LIFO schedule, and the FIFO schedule with idle time in p_2.
These schedules are referred to as Schedule f, Schedule l, and Schedule g respectively. Superscripts f, l, and g are used to distinguish the three schedules. The equations for load fractions, processing times, and the conditions for optimality of Schedules f, l, and g are not derived on account of space constraints. The interested reader is directed to (Ghatpande, Nakazato, Beaumont & Watanabe, 2008) for details.

4.1 Optimal Schedule in Two-Slave System

A few lemmas and theorems to determine the optimal schedule for a two-slave system are now stated without proof. Please refer to Ghatpande, Nakazato, Beaumont & Watanabe (2008) for the proofs.

Lemma 4. *It is always advantageous to distribute the load to both the processors, rather than execute it on the individual processors (for the system model under consideration).*

Lemma 5 (Idle Indicator Lemma)**.** *$\rho_1\rho_2 \leq \delta$ is a necessary and sufficient condition to indicate the presence of idle time in the FIFO schedule (i.e. Schedule g).*

The simplicity of the condition to detect the presence of idle time in the FIFO schedule is both pleasing and surprising, and has been derived for the first time ever. Further confirmation of this condition is obtained in Sect. 4.2.

Theorem 3 (Optimal Schedule Theorem)**.** *The optimal schedule for a two-slave system can be found as follows:*

1. *If $\delta C_2 > C_1(1 + \delta + \rho_1)$, then Schedule l is optimal.*
2. *Else If $\delta C_2 \leq C_1(1 + \delta + \rho_1)$, $\rho_1\rho_2 \leq \delta$ and $C_2 \leq C_1\left(1 + \frac{(1+\rho_1)\rho_2}{\delta(1+\delta+\rho_2)}\right)$, then Schedule g is optimal.*
3. *Else if $\delta C_2 \leq C_1(1 + \delta + \rho_1)$, $\rho_1\rho_2 \leq \delta$ and $C_2 > C_1\left(1 + \frac{(1+\rho_1)\rho_2}{\delta(1+\delta+\rho_2)}\right)$, then Schedule l is optimal.*
4. *Else If $\delta C_2 \leq C_1(1 + \delta + \rho_1)$, $\rho_1\rho_2 > \delta$, and $T^f \leq \frac{C_1C_2}{(C_2-C_1)}$, then Schedule f is optimal.*
5. *Else if $\delta C_2 \leq C_1(1 + \delta + \rho_1)$, $\rho_1\rho_2 > \delta$, and $T^f > \frac{C_1C_2}{(C_2-C_1)}$, then Schedule l is optimal.*

The optimal solution to DLSRCHETS, $(\sigma_a^*, \sigma_c^*, \alpha^*)$, for a system with two slave processors is a function of the system parameters and the application under consideration, because of which, no particular sequence of allocation and collection can be defined *a priori* as the optimal sequence. The optimal solution can only be determined once all the parameters become known.

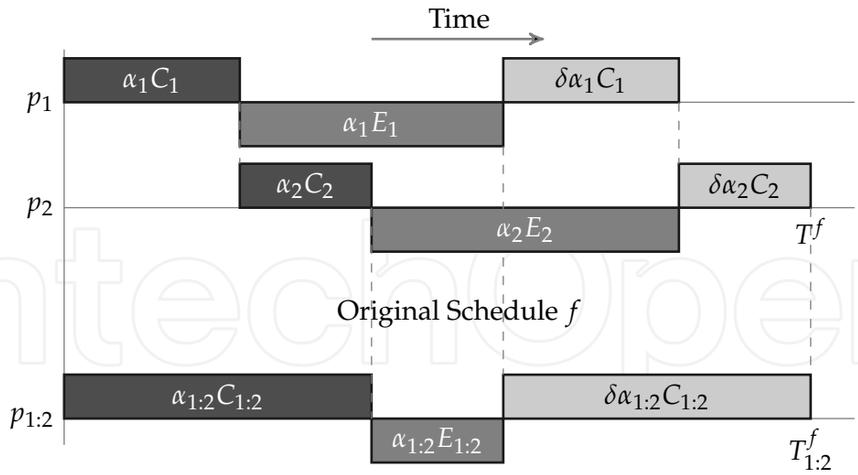

Fig. 7. Equivalent processor in Schedule f. The total communication time remains the same as the original two processors. The equivalent computation time is equal to the interval between the end of allocation to p_2 and the start of result collection from p_1.

4.2 The Concept of Equivalent Processor

To extend the above result to the general case with m slave processors, the concept of an *equivalent processor* is introduced. Consider the system in Fig. 6. The processors p_1 and p_2 are replaced by a single equivalent processor $p_{1:2}$ with computation parameter $E_{1:2}$, connected to the root by an equivalent link $l_{1:2}$ with communication parameter $C_{1:2}$. The resulting system is called the *equivalent system* and the resulting schedule is known as the *equivalent schedule*. The values of the parameters for the three equivalent schedules are defined below.
If the initial load distribution is $\alpha = \{\alpha_1, \alpha_2\}$, and the processing time is T, then the equivalent system satisfies the following properties:

- The load processed by $p_{1:2}$ is $\alpha_{1:2} = \alpha_1 + \alpha_2 = 1$.
- The processing time is unchanged and equal to T.
- The time spent in load distribution and result collection is unchanged, i.e., for all three schedules,
  - $\alpha_{1:2} C_{1:2} = \alpha_1 C_1 + \alpha_2 C_2$, and
  - $\delta\alpha_{1:2} C_{1:2} = \delta\alpha_1 C_1 + \delta\alpha_2 C_2$.
- The time spent in load computation is equal to the intervening time interval between the end of allocation phase and the start of result collection phase, i.e.,
  - For Schedule f, $\alpha_{1:2} E^f_{1:2} = \alpha_1 E_1 - \alpha_2 C_2 = \alpha_2 E_2 - \delta\alpha_1 C_1$.
  - For Schedule l, $\alpha_{1:2} E^l_{1:2} = \alpha_2 E_2 = \alpha_1 E_1 - \alpha_2 C_2 - \delta\alpha_2 C_2$.
  - For Schedule g, $\alpha_{1:2} E^g_{1:2} = 0$.

4.3 The Equivalent Processor Theorem

This leads to the following theorem: (refer to (Ghatpande, Nakazato, Beaumont & Watanabe, 2008) for proof.)

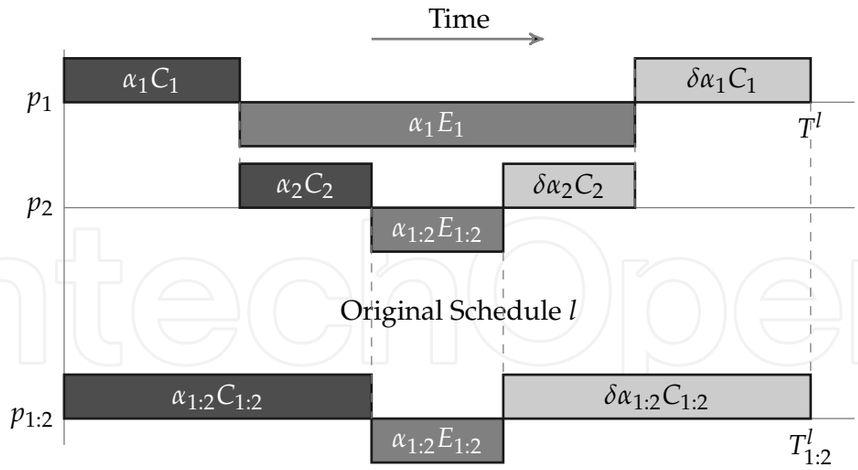

Fig. 8. Equivalent processor in Schedule l. The total communication time remains the same as the original two processors. The equivalent computation time is equal to the computation time of p_2.

Theorem 4 (Equivalent Processor Theorem)**.** *In a heterogeneous system $\mathcal{H}$ with $m = 2$, the two slave processors p_1 and p_2 can be replaced without affecting the processing time T, by a single (virtual) equivalent processor $p_{1:2}$ with equivalent parameters $C_{1:2}$ and $E_{1:2}$, such that $C_1 \leq C_{1:2} \leq C_2$ and $E_{1:2} \leq E_1, E_2$.*

The equivalent processor enables replacement of two processors by a single processor with communication parameter with a value that lies between the values of communication parameters of the original two links. Because of this property, if the processors are arranged so that $C_1 \leq C_2 \leq \ldots \leq C_m$, and two processors are combined at a time sequentially starting from the fastest two, then the resultant equivalent processor does not disturb the order of the sequence.
The equivalent processor for Schedule f provides additional confirmation of the condition for the presence of idle time in a FIFO schedule. It is known that idle time can exist in a FIFO schedule only when the intervening time interval $y = 0$. According to the definition of equivalent processor, this interval corresponds to the equivalent computation capacity $E^f_{1:2}$. This value becomes zero only when $\rho_1\rho_2 - \delta = 0$. Thus, if $\rho_1\rho_2 < \delta$, then idle time must exist in the FIFO schedule.

5. The SPORT Algorithm

Algorithm 1 (SPORT)**.**

1: *arrange $p_1, \ldots, p_m$ such that $C_1 \leq C_2 \leq \ldots \leq C_m$*

2: $\sigma_a \leftarrow 1$, $\sigma_c \leftarrow 1$, $\alpha_1 \leftarrow 1$

3: **for** $k := 2$ *to* m **do**

4: $C_1 \leftarrow C_{1:k-1}$, $E_1 \leftarrow E_{1:k-1}$, $C_2 \leftarrow C_k$, $E_2 \leftarrow E_k$

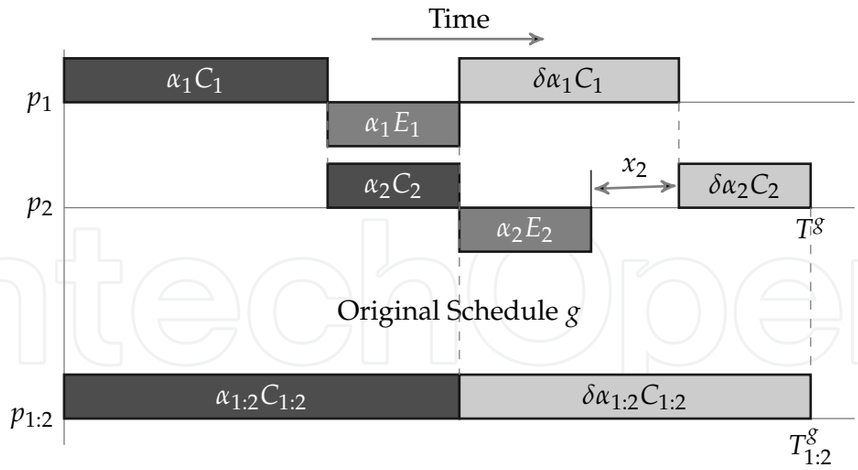

Fig. 9. Equivalent processor in Schedule g. The total communication time remains the same as the original two processors. The equivalent computation time is equal to zero as the result collection begins immediately after the allocation phase ends.

5: **if** $\delta C_2 > C_1(1 + \delta + \rho_1)$ **then**

6: */* $T^l < T^f, T^g$, use Schedule l */*

7: **call** `schedule_lifo`

8: **else**

9: */* Need to check other conditions */*

10: **if** $\rho_1\rho_2 \leq \delta$ **then**

11: */* Possibility of idle time */*

12: **if** $C_2 \leq C_1\left(1 + \frac{(1+\rho_1)\rho_2}{\delta(1+\delta+\rho_2)}\right)$ **then**

13: */* $T^g < T^l$, use Schedule g */*

14: **call** `schedule_idle`

15: **break for**

16: **else**

17: */* $T^l < T^g$, use Schedule l */*

18: **call** `schedule_lifo`

19: **end if**

20: **else**

21: */* No idle time present */*

22: **if** $T^f \leq \dfrac{C_1 C_2}{C_2 - C_1}$ **then**

23: */* $T^f < T^l$, use Schedule f */*

24: **call** `schedule_fifo`

25: **else**

26: */* $T^l < T^f$, use Schedule l */*

27: **call** `schedule_lifo`

28: **end if**

29: **end if**

30: **end if**

31: **end for**

32: $n \leftarrow$ `numberOfProcessorsUsed`

33: */* Update load fractions from stored values */*

34: $\alpha_k \leftarrow \begin{cases} \alpha_k \cdot \prod_{j=2}^{n} \alpha_{1:j} & \text{if } k = 1 \\ \alpha_k \cdot \prod_{j=k}^{n} \alpha_{1:j} & \text{if } k = 2, \ldots, n \end{cases}$

35: $T \leftarrow C_{1:n} + E_{1:n} + \delta C_{1:n}$

The procedures in the algorithm are given below:

procedure `schedule_idle`

1: $\alpha_{1:k-1} \leftarrow \dfrac{C_2}{C_1\rho_1 + C_2}$

2: $\alpha_k \leftarrow \dfrac{C_1\rho_1}{C_1\rho_1 + C_2}$

3: */* Update sequences for FIFO */*

4: $\sigma_a \leftarrow \{\sigma_a,\ k\}$

5: $\sigma_c \leftarrow \{\sigma_c,\ k\}$

6: */* Compute equivalent processor parameters */*

7: $C_{1:k} \leftarrow \dfrac{C_1 C_2 (1 + \rho_1)}{C_1\rho_1 + C_2}$

8: $E_{1:k} \leftarrow 0$

9: `numberOfProcessorsUsed` $\leftarrow k$

10: **return**

procedure `schedule_lifo`

1: $r_1^l \leftarrow \rho_1$

2: $r_2^l \leftarrow 1 + \delta + \rho_2$

3: $\alpha_{1:k-1} \leftarrow \dfrac{C_2 r_2^l}{C_1 r_1^l + C_2 r_2^l}$

4: $\alpha_k \leftarrow \dfrac{C_1 r_1^l}{C_1 r_1^l + C_2 r_2^l}$

5: */* Update sequences for LIFO */*

6: $\sigma_a \leftarrow \{\sigma_a,\ k\}$

7: $\sigma_c \leftarrow \{k,\ \sigma_c\}$

8: */* Compute equivalent processor parameters */*

9: $C_{1:k} \leftarrow \dfrac{C_1 C_2 (r_1^l + r_2^l)}{C_1 r_1^l + C_2 r_2^l}$

10: $E_{1:k} \leftarrow \dfrac{C_1 C_2 \rho_1 \rho_2}{C_1 r_1^l + C_2 r_2^l}$

11: `numberOfProcessorsUsed` $\leftarrow k$

12: **return**

procedure `schedule_fifo`

1: $r_1^f \leftarrow \delta + \rho_1$

2: $r_2^f \leftarrow 1 + \rho_2$

3: $\alpha_{1:k-1} \leftarrow \dfrac{C_2 r_2^f}{C_1 r_1^f + C_2 r_2^f}$

4: $\alpha_k \leftarrow \dfrac{C_1 r_1^f}{C_1 r_1^f + C_2 r_2^f}$

5: */* Update sequences for FIFO */*

6: $\sigma_a \leftarrow \{\sigma_a,\ k\}$

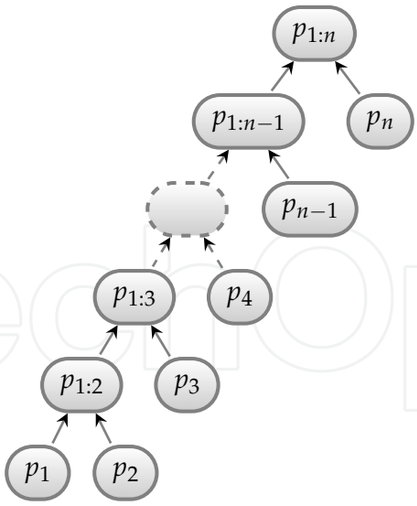

Fig. 10. The building of SPORT solution. At each step only two processors are involved (the state space remains constant). The optimal schedule for two processors can be easily computed in constant time using simple if-then-else statements in Theorem 3.

7: $\sigma_c \leftarrow \{\sigma_c, k\}$

8: */* Compute equivalent processor parameters */*

9: $C_{1:k} \leftarrow \dfrac{C_1 C_2 (r_1^f + r_2^f)}{C_1 r_1^f + C_2 r_2^f}$

10: $E_{1:k} \leftarrow \dfrac{C_1 C_2 (\rho_1 \rho_2 - \delta)}{C_1 r_1^f + C_2 r_2^f}$

11: `numberOfProcessorsUsed` $\leftarrow k$

12: **return**

5.1 Algorithm Explanation

At the start, the processors are arranged so that $C_1 \leq C_2 \leq \ldots \leq C_m$, and two processors with the fastest communication links are selected. The optimal schedule and load distribution for the two processors are found according to Theorem 3. If Schedule f or l is found optimal, then the two processors are replaced by their equivalent processor. In either case, since $C_1 \leq C_{1:2} \leq C_2$, the ordering of the processors does not change. In the subsequent iteration, the equivalent processor and the processor with the next fastest communication link are selected and the steps are repeated until either all processors are used up, or Schedule g is found to be optimal. If Schedule g is found to be optimal in any iteration, then the algorithm exits after finding the load distribution for that iteration.

The computation of the allocation and collection sequences is straightforward. The allocation sequence σ_a is maintained in the order of decreasing communication link bandwidth of the processors. Irrespective of the schedule found optimal in iteration k, k is always appended to σ_a. The collection sequence σ_c is constructed as follows:

- If Schedule f or g is found optimal in iteration k, k is appended to σ_c.

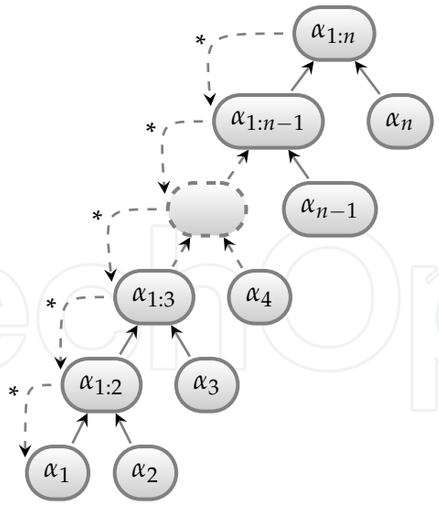

Fig. 11. Calculating the load fractions in SPORT. α'_1 is the initial value of α_1. It is multiplied by the product term in (20) to get the final value of $\alpha_1 = \alpha_{1:n} \cdot \alpha_{1:n-1} \cdots \alpha_{1:2} \cdot \alpha'_1$. This is equivalent to traversing the binary tree from the root to the leaf nodes and taking the product of all nodes (values) encountered. This calculation can be implemented in $O(m)$ time by starting with α_m and storing the intermediate values.

- If Schedule l is found optimal in iteration k, k is prepended to σ_c.

The calculation of load distribution to the processors occurs simultaneously with the search for the optimal schedule. As shown in Fig. 11, the algorithm creates a *one-sided binary tree* of load fractions. If the number of processors participating in the computation is n, $2 \leq n \leq m$, the root node of the binary tree is $\alpha_{1:n}$ and the leaf nodes represent the final load fractions allocated to the processors. The value of the root node need not be calculated as it is equal to one. The individual load fractions, α_k, are initially assigned value α'_k (say), and then updated at the end as:

$$\alpha_k = \begin{cases} \alpha'_k \cdot \prod_{j=2}^{n} \alpha_{1:j} & \text{if } k = 1 \\ \alpha'_k \cdot \prod_{j=k}^{n} \alpha_{1:j} & \text{if } k = 2, \ldots, n \end{cases} \tag{20}$$

This is equivalent to traversing the binary tree from the root to each leaf node and taking the product of the nodes encountered (see Fig. 11). This calculation can be easily implemented in $O(m)$ time by starting with the computation of α_n, and storing the values of the product terms (i.e. $\prod \alpha_{1:j}$) for each processor and then using that value for the next processor.
Once the sequences (σ_a, σ_c) and load distribution α are found, calculating the processing time is straightforward. The processing time is simply the sum of the (equivalent) parameters of the equivalent processor $p_{1:n}$, i.e., $T = C_{1:n} + E_{1:n} + \delta C_{1:n}$.
In SPORT, defining the allocation sequence by sorting the values of C_k requires $O(m \log m)$ time, while finding the collection sequence and load distribution requires $O(m)$ time in the worst case. Thus, if sorted values of C_k are given, then the overall complexity of the algorithm is polynomial in m and is equal to $O(m)$.

5.2 Simulations and Analysis

The performance of SPORT was compared to four algorithms, viz. OPT, FIFOC, LIFOC, and ITERLP. The globally optimal schedule OPT is obtained after evaluation of the linear pro-

Table 1. Minimum statistics for SPORT simulations. In sets 1 and 2, the minimum errors in LIFOC are 2 orders of magnitude higher than SPORT, ITERLP, and FIFOC. In sets 3 and 4, FIFOC error is 2 to 3 orders of magnitude higher than the other three algorithms.

Set	m	$\delta = 0.2$				$\delta = 0.5$			
		SPORT	ITERLP	LIFOC	FIFOC	SPORT	ITERLP	LIFOC	FIFOC
1	4	5.73e-03	4.32e-03	8.08e-01	5.76e-03	2.20e-02	1.06e-02	1.07e+00	2.21e-02
	5	7.89e-04	6.90e-04	7.21e-01	7.89e-04	5.40e-03	4.21e-03	9.63e-01	5.30e-03
2	4	1.01e-02	5.78e-03	8.41e-01	1.01e-02	2.37e-02	1.43e-02	1.15e+00	2.40e-02
	5	3.34e-03	2.10e-03	7.93e-01	3.34e-03	1.06e-02	8.92e-03	1.10e+00	1.07e-02
3	4	2.03e-01	1.80e-03	1.05e-01	1.61e+00	1.12e-01	5.13e-03	9.59e-02	4.43e+00
	5	3.96e-01	1.90e-01	8.90e-02	1.75e+00	5.34e-02	9.32e-02	5.13e-02	4.74e+00
4	4	4.95e-06	1.97e-16	4.92e-06	1.05e+00	3.09e-02	2.77e-15	3.09e-02	3.23e+00
	5	1.08e-02	5.81e-04	2.75e-06	1.15e+00	5.84e-02	2.18e-03	5.84e-02	3.74e+00

Table 2. Maximum statistics for SPORT simulations. In sets 1 and 2, the maximum errors in LIFOC are 2 orders of magnitude higher than SPORT, ITERLP, and FIFOC. In sets 3 and 4, FIFOC error is 2 to 3 orders of magnitude higher than the other three algorithms.

Set	m	$\delta = 0.2$				$\delta = 0.5$			
		SPORT	ITERLP	LIFOC	FIFOC	SPORT	ITERLP	LIFOC	FIFOC
1	4	5.34e-02	3.09e-02	3.11e+00	5.61e-02	1.84e-01	7.57e-02	4.20e+00	2.02e-01
	5	8.24e-02	4.87e-02	3.00e+00	8.79e-02	2.26e-01	1.19e-01	3.91e+00	2.30e-01
2	4	3.03e-02	1.69e-02	1.83e+00	3.06e-02	9.35e-02	4.93e-02	3.10e+00	1.10e-01
	5	3.66e-02	2.61e-02	2.24e+00	3.68e-02	1.15e-01	8.34e-02	2.75e+00	1.26e-01
3	4	4.01e-01	3.42e-01	4.66e-01	2.02e+00	4.03e-01	2.22e-01	4.03e-01	5.44e+00
	5	5.31e-01	3.86e-01	4.84e-01	2.30e+00	5.45e-01	3.80e-01	4.16e-01	6.05e+00
4	4	1.32e+00	6.50e-01	8.84e-01	4.47e+00	8.02e-01	7.11e-01	4.00e-01	1.12e+01
	5	1.56e+00	7.66e-01	4.34e-01	4.85e+00	9.35e-01	8.97e-01	4.24e-01	1.15e+01

gram for all possible $(m!)^2$ permutations of (σ_a, σ_c). In FIFOC, processors are allocated load and result are collected in the order of decreasing communication link bandwidth of the processors. In LIFOC, load allocation is in the order of decreasing communication link bandwidth of the processors, while result collection is the reverse order of increasing communication link bandwidth of the processors. ITERLP (Ghatpande, Beaumont, Nakazato & Watanabe, 2008) is a near-optimal algorithm for DLSRCHETS. To explore the effects of system parameter values on the performance of the algorithms, several sets of simulations were carried out:

Set 1 Homogeneous network and homogeneous processors
Set 2 Homogeneous network and heterogeneous processors
Set 3 Heterogeneous network and homogeneous processors
Set 4 Heterogeneous network and heterogeneous processors

The error values with respective to the optimal are calculated. Over 500,000 simulation runs are carried out. Further details can be obtained in (Ghatpande, Beaumont, Nakazato & Watanabe, 2008; Ghatpande, Nakazato, Beaumont & Watanabe, 2008). The minimum and maximum

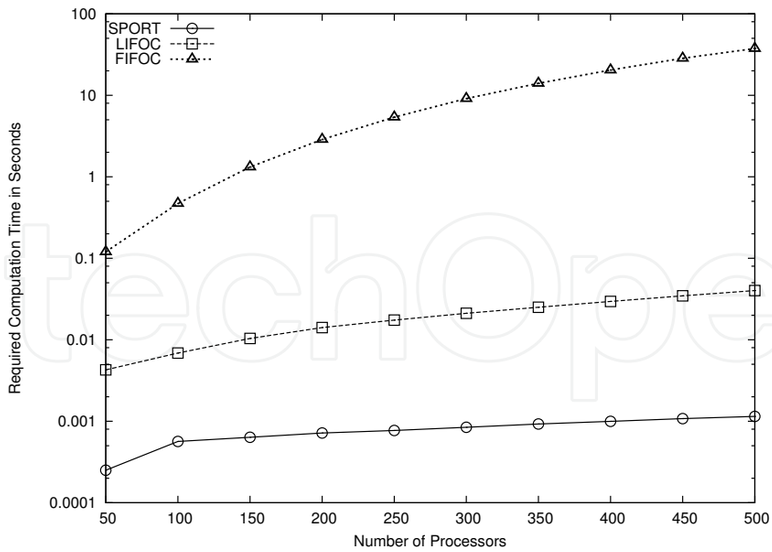

Fig. 12. Comparison of wall-clock time for SPORT, LIFOC, and FIFOC. SPORT is two orders of magnitude faster than LIFOC and almost four orders of magnitude faster than FIFOC. This figure appears in (Ghatpande, Nakazato, Beaumont & Watanabe, 2008).

mean error values of each algorithm are tabulated in Tables 1 and 2. It can be observed that in sets 1 and 2, the minimum and maximum errors in LIFOC are 2 orders of magnitude higher than SPORT, ITERLP, and FIFOC. On the other hand in sets 3 and 4, FIFOC error is 2 to 3 orders of magnitude higher than the other three algorithms.
There is a significant downside to LIFOC because of its property to use all available processors — the time required to compute the optimal solution (wall-clock time) is almost two orders of magnitude greater than that of SPORT as seen in Fig. 12. These values were obtained by averaging the wall-clock time to compute a solution over 1000 runs. The results show that though both SPORT and LIFOC are $O(m)$ algorithms given a set of processors sorted by decreasing communication bandwidth, clearly SPORT is the better performing algorithm, with the best cost-performance ratio for large values of m. The values for FIFOC are almost four orders of magnitude larger than SPORT. The extensive simulations show that:

- If network links are homogeneous, then LIFOC performance is affected for both homogeneous and heterogeneous computation speeds.
- If network links are heterogeneous, then FIFOC performance is affected for both homogeneous and heterogeneous computation speeds.
- SPORT performance is also affected to a certain degree by the heterogeneity in network links and computation speeds, but since SPORT does not use a single predefined sequence of allocation and collection, it is able to better adapt to the changing system conditions.
- ITERLP performance is somewhat better than SPORT, but is computationally expensive. SPORT generates similar schedules at a fraction of the cost.

6. Conclusion

In this chapter, the DLSRCHETS problem for the scheduling of divisible loads on heterogeneous master-slave systems and considering the result collection phase was formulated and

analysed. A new polynomial-time algorithm, SPORT was proposed and tested. Future work can proceed in the following main directions:

Theoretical Analysis The complexity of DLSRCHETS is still an open issue. It makes for an interesting research topic. Is it at all possible that DLSRCHETS can be solved in polynomial time? Does imposition of some additional constraints make it tractable? What are those conditions?

Extending the System Model This area has a large number of possibilities for future work. Scheduling purists may consider the system model used in this thesis to be quite simplistic. As future work, the conditions (constraints on values of E_k and C_k), that minimize the error need to be found. An interesting area would be the investigation of the effect of affine cost models, processor deadlines and release times. Another important area would be to extend the results to multi-installment delivery and multi-level processor trees.

Modification of DLSRCHETS The ways in which DLSRCHETS may be modified are — dynamism and uncertainty in the system parameters, non-clairvoyance, non-omniscience of the master, node (slave) turnover (failure), slave sharing, multiple jobs on one master, multiple masters, multiple jobs on several masters, decentralization of scheduling decision (P2P model), QoS requirements, buffer, bandwidth, and computation constraints on slaves.

Application Development All the testing in this work has been carried out using simulations. It will be interesting to see how the algorithms perform in practice. New and different applications apart from the number of possible scientific applications mentioned in the introduction, need to be developed that use the results in this work. This may require development of new libraries and middleware to support the computation models considered.

7. References

Adler, M., Gong, Y. & Rosenberg, A. L. (2003). Optimal sharing of bags of tasks in heterogeneous clusters, *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, ACM, New York, NY, USA, pp. 1–10.

Barlas, G. D. (1998). Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees, **9**(5): 429–441.

Beaumont, O., Casanova, H., Legrand, A., Robert, Y. & Yang, Y. (2005). Scheduling divisible loads on star and tree networks: Results and open problems, **16**(3): 207–218.

Beaumont, O., Marchal, L., Rehn, V. & Robert, Y. (2005). FIFO scheduling of divisible loads with return messages under the one-port model, *Research Report 2005-52*, LIP, ENS Lyon, France.

Beaumont, O., Marchal, L., Rehn, V. & Robert, Y. (2006). FIFO scheduling of divisible loads with return messages under the one port model, *Proc. Heterogeneous Computing Workshop HCW'06*.

Beaumont, O., Marchal, L. & Robert, Y. (2005). Scheduling divisible loads with return messages on heterogeneous master-worker platforms, *Research Report 2005-21*, LIP, ENS Lyon, France.

Bharadwaj, V., Ghose, D., Mani, V. & Robertazzi, T. G. (1996). *Scheduling Divisible Loads in Parallel and Distributed Systems*, IEEE Computer Society Press, Los Alamitos, CA.

Cheng, Y.-C. & Robertazzi, T. G. (1990). Distributed computation for a tree network with communication delays, **26**(3): 511–516.

Comino, N. & Narasimhan, V. L. (2002). A novel data distribution technique for host-client type parallel applications, **13**(2): 97–110.

Dantzig, G. B. (1963). *Linear Programming and Extensions*, Princeton Univ. Press, Princeton, NJ.

Ghatpande, A., Beaumont, O., Nakazato, H. & Watanabe, H. (2008). Divisible load scheduling with result collection on heterogeneous systems, *Proc. Heterogeneous Computing Workshop (HCW 2008) held in the IEEE Intl. Parallel and Distributed Processing Sysmposium (IPDPS 2008)*, Miami, FL.

Ghatpande, A., Nakazato, H., Beaumont, O. & Watanabe, H. (2008). SPORT: An algorithm for divisible load scheduling with result collection on heterogeneous systems, *IEICE Transactions on Communications* **E91-B**(8).

Robertazzi, T. (2008). Divisible (partitionable) load scheduling research.
URL: *http://www.ece.sunysb.edu/ tom/dlt.html#THEORY*

Rosenberg, A. (2001). Sharing partitionable workload in heterogeneous NOWs: Greedier is not better, *IEEE International Conference on Cluster Computing*, Newport Beach, CA, pp. 124–131.

Vanderbei, R. J. (2001). *Linear Programming: Foundations and Extensions*, Vol. 37 of *International Series in Operations Research & Management*, 2nd edn, Kluwer Academic Publishers.
URL: *http://www.princeton.edu/ rvdb/LPbook/online.html*

Yu, D. & Robertazzi, T. G. (2003). Divisible load scheduling for grid computing, *Proc. International Conference on Parallel and Distributed Computing Systems (PDCS 2003)*, Vol. 1, Los Angeles, CA, USA.

Parallel and Distributed Computing

Edited by Alberto Ros

ISBN 978-953-307-057-5

Hard cover, 290 pages

Publisher InTech

Published online 01, January, 2010

Published in print edition January, 2010

The 14 chapters presented in this book cover a wide variety of representative works ranging from hardware design to application development. Particularly, the topics that are addressed are programmable and reconfigurable devices and systems, dependability of GPUs (General Purpose Units), network topologies, cache coherence protocols, resource allocation, scheduling algorithms, peertopeer networks, largescale network simulation, and parallel routines and algorithms. In this way, the articles included in this book constitute an excellent reference for engineers and researchers who have particular interests in each of these topics in parallel and distributed computing.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen