Scheduling strategies for mixed data and task parallelism on heterogeneous clusters and grids

O. Beaumont
LaBRI, UMR CNRS 5800
Bordeaux, France
Olivier.Beaumont@labri.fr

A. Legrand and Y. Robert
LIP, UMR CNRS-INRIA 5668
ENS Lyon, France
{Arnaud.Legrand,Yves.Robert}@ens-lyon.fr

Abstract

We consider the execution of a complex application on a heterogeneous "grid" computing platform. The complex application consists of a suite of identical, independent problems to be solved. In turn, each problem consists of a set of tasks. There are dependences (precedence constraints) between these tasks. A typical example is the repeated execution of the same algorithm on several distinct data samples. We use a non-oriented graph to model the grid platform, where resources have different speeds of computation and communication. We show how to determine the optimal steady-state scheduling strategy for each processor (the fraction of time spent computing and the fraction of time spent communicating with each neighbor). This result holds for a quite general framework, allowing for cycles and multiple paths in the platform graph.

1 Introduction

In this paper, we consider the execution of a complex application, on a heterogeneous "grid" computing platform. The complex application consists of a suite of identical, independent problems to be solved. In turn, each problem consists of a set of tasks. There are dependences (precedence constraints) between these tasks. A typical example is the repeated execution of the same algorithm on several distinct data samples. Consider the simple fork graph depicted in Figure 1. This fork graph models the algorithm. We borrow this example from Subhlok et al. [30]. There is a main loop which is executed several times. Within each loop iteration, there are four tasks to be performed on some matrices. Each loop iteration is what we call a problem instance. Each problem instance operates on different data, but all instances share the same *task graph*, i.e. the fork graph of Figure 1. For each node in the task graph, there are as many task copies as there are iterations in the main loop.

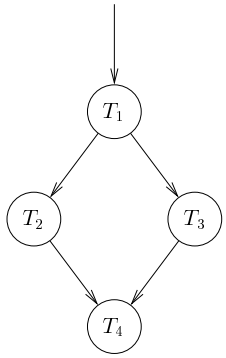

Figure 1. A simple fork graph example.

We use another graph, the *platform graph*, for the grid platform. We model a collection of heterogeneous resources and the communication links between them as the nodes and edges of an undirected graph. See the example in Figure 2 with four processors and five communication links. Each node is a computing resource (a processor, or a cluster, or whatever) capable of computing and/or communicating with its neighbors at (possibly) different rates. The underlying interconnection network may be very complex and, in particular, may include multiple paths and cycles (just as the Ethernet does).

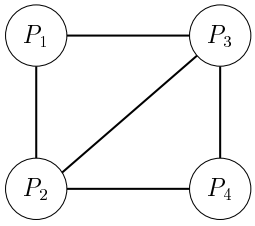

Figure 2. A simple platform example

We assume that one specific node, referred to as the master, initially holds (or generates the data for) the input tasks

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

of all problems. The question for the master is to decide which tasks to execute itself, and how many tasks to forward to each of its neighbors. Due to heterogeneity, the neighbors may receive different amounts of work (maybe none for some of them). Each neighbor faces in turn the same dilemma: determine how many tasks to execute, and how many to delegate to other processors. Note that the master may well need to send tasks along multiple paths to properly feed a very fast but remote computing resource.

Because the problems are independent, their execution can be pipelined. At a given time-step, different processors may well compute different tasks belonging to different problem instances. In the example, a given processor P_i may well compute the tenth copy of task T_1, corresponding to problem number 10, while another processor P_j computes the eight copy of task T_3, which corresponds to problem number 8. However, because of the dependence constraints, note that P_j could not begin the execution of the tenth copy of task T_3 before that P_i has terminated the execution of the tenth copy of task T_1 and sent the required data to P_j (if $i \neq j$).

In this paper, our objective is to determine the optimal steady state scheduling policy for each processor, i.e. the fraction of time spent computing, and the fraction of time spent sending or receiving each type of tasks along each communication link, so that the (averaged) overall number of problems processed at each time-step is maximum.

This scheduling problem is motivated by problems that are addressed by collaborative computing efforts such as SETI@home [22], factoring large numbers [10], the Mersenne prime search [20], and those distributed computing problems organized by companies such as Entropia [11]. Several papers [26, 25, 13, 12, 34, 5, 4] have recently revisited the master-slave paradigm for processor clusters or grids, but all these papers only deal with independent tasks. To the best of our knowledge, the algorithm presented in this paper is the first that allows precedence constraints in a heterogeneous framework. In other words, this paper represents a first step towards extending all the work on mixed task and data parallelism [30, 8, 21, 1, 31] towards heterogeneous platforms.

The rest of the paper is organized as follows. In Section 2, we introduce our base model of computation and communication, and we formally state the steady-state scheduling to be solved. In Section 3, we provide the optimal solution to this problem, using a linear programming approach. We work out a full example in Section 4. We briefly survey related work in Section 5. Finally, we give some remarks and conclusions in Section 6.

2 The model

We start with a formal description of the application/architecture framework. Next we state all the equations that hold during steady-state operation.

2.1 Application/architecture framework

The application

- Let $\mathcal{P}^{(1)}, \mathcal{P}^{(2)}, \ldots, \mathcal{P}^{(n)}$ be the n problems to solve, where n is large
- Each problem $\mathcal{P}^{(m)}$ corresponds to a copy $G^{(m)} = (V^{(m)}, E^{(m)})$ of the same *task graph* (V, E). The number $|V|$ of nodes in V is the number of task types. In the example of Figure 1, there are four task types, denoted as T_1, T_2, T_3 and T_4.
- Overall, there are $n.|V|$ tasks to process, since there are n copies of each task type.

The architecture

- The target heterogeneous platform is represented by a directed graph, the *platform graph*.
- There are p nodes $P_1, P_2, \ldots, P_p$ that represent the processors. In the example of Figure 2 there are four processors, hence $p = 4$. See below for processor speeds and execution times.
- Each edge represents a physical interconnection. Each edge $e_{ij} : P_i \to P_j$ is labeled by a value c_{ij} which represents the time to transfer a message of unit length between P_i and P_j, in either direction: we assume that the link between P_i and P_j is bidirectional and symmetric. A variant would be to assume two unidirectional links, one in each direction, with possibly different label values. If there is no communication link between P_i and P_j we let $c_{ij} = +\infty$, so that $c_{ij} < +\infty$ means that P_i and P_j are neighbors in the communication graph. With this convention, we can assume that the interconnection graph is (virtually) complete.
- We assume a *full overlap, single-port* operation mode, where a processor node can simultaneously receive data from one of its neighbor, perform some (independent) computation, and send data to one of its neighbor. At any given time-step, there are at most two communications involving a given processor, one in emission and the other in reception. Other models can be dealt with, see [4, 2].

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

Execution times

- Processor P_i requires $w_{i,k}$ time units to process a task of type T_k.
- Note that this framework is quite general, because each processor has a different speed for each task type, and these speeds are not related: they are *inconsistent* with the terminology of [7]. Of course, we can always simplify the model. For instance we can assume that $w_{i,k} = w_i \times \delta_k$, where w_i is the inverse of the relative speed of processor P_i, and δ_k the weight of task T_k.

Communication times

- Each edge $e_{k,l} : T_k \to T_l$ in the task graph is weighted by a communication cost $data_{k,l}$ that depends on the tasks T_k and T_l. It corresponds to the amount of data output by T_k and required as input to T_l.
- Recall that the time needed to transfer a unit amount of data from processor P_i to processor P_j is $c_{i,j}$. Thus, if a task $T_k^{(m)}$ is processed on P_i and task $T_l^{(m)}$ is processed on P_j, the time to transfer the data from P_i to P_j is equal to $data_{k,l} \times c_{i,j}$; this holds for any edge $e_{k,l} : T_k \to T_l$ in the task graph and for any processor pair P_i and P_j. Again, once a communication from P_i to P_j is initiated, P_i (resp. P_j) cannot handle a new emission (resp. reception) during the next $data_{k,l} \times c_{i,j}$ time units.

2.2 Steady-state equations

We begin with a few definitions:

- For each edge $e_{k,l} : T_k \to T_l$ in the task graph and for each processor pair (P_i, P_j), we denote by $s(P_i \to P_j, e_{k,l})$ the (average) fraction of time spent each time-unit by P_i to send to P_j data involved by the edge $e_{k,l}$. Of course $s(P_i \to P_j, e_{k,l})$ is a non-negative rational number. Think of an edge $e_{k,l}$ as requiring a new file to be transferred from the output of each task $T_k^{(m)}$ processed on P_i to the input of each task $T_l^{(m)}$ processed on P_j. Let the (fractional) number of such files sent per time-unit be denoted as $sent(P_i \to P_j, e_{k,l})$. We have the relation:

$$s(P_i \to P_j, e_{k,l}) = sent(P_i \to P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) \quad (1)$$

  which states that the fraction of time spent transferring such files is equal to the number of files times the product of their size by the elemental transfer time of the communication link.

- For each task type $T_k \in V$ and for each processor P_i, we denote by $\alpha(P_i, T_k)$ the (average) fraction of time spent each time-unit by P_i to process tasks of type T_k, and by $cons(P_i, T_k)$ the (fractional) number of tasks of type T_k processed per time unit by processor P_i. We have the relation

$$\alpha(P_i, T_k) = cons(P_i, T_k) \times w_{i,k} \quad (2)$$

We search for rational values of all the variables $s(P_i \to P_j, e_{k,l})$, $sent(P_i \to P_j, e_{k,l})$, $\alpha(P_i, T_k)$ and $cons(P_i, T_k)$. We formally state the first constraints to be fulfilled.

Activities during one time-unit All fractions of time spent by a processor to do something (either computing or communicating) must belong to the interval $[0, 1]$, as they correspond to the average activity during one time unit:

$$\forall P_i, \forall T_k \in V,\ 0 \le \alpha(P_i, T_k) \le 1 \quad (3)$$

$$\forall P_i, P_j, \forall e_{k,l} \in E,\ 0 \le s(P_i \to P_j, e_{k,l}) \le 1 \quad (4)$$

One-port model for outgoing communications Because send operations to the neighbors of P_i are assumed to be sequential, we have the equation:

$$\forall P_i, \sum_{P_j \in n(P_i)} \sum_{e_{k,l} \in E} s(P_i \to P_j, e_{k,l}) \le 1 \quad (5)$$

where $n(P_i)$ denotes the neighbors of P_i. Recall that we can assume a complete graph owing to our convention with the $c_{i,j}$.

One-port model for incoming communications Because receive operations from the neighbors of P_i are assumed to be sequential, we have the equation:

$$\forall P_i, \sum_{P_j \in n(P_i)} \sum_{e_{k,l} \in E} s(P_j \to P_i, e_{k,l}) \le 1 \quad (6)$$

Note that $s(P_j \to P_i, e_{k,l})$ is indeed equal to the fraction of time spent by P_i to receive from P_j files of type $e_{k,l}$.

Full overlap because of the full overlap hypothesis, there is no further constraint on $\alpha(P_i, T_k)$ except that

$$\forall P_i, \sum_{T_k \in V} \alpha(P_i, T_k) \le 1 \quad (7)$$

For technical reasons it is simpler to have a single input task (a task without any predecessor) and a single output task (a task without any successor) in the task graph. To this purpose, we introduce two fictitious tasks, T_{begin} which is connected to every task with no predecessor in the graph,

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

and T_{end} which is connected to every task with no successor in the graph. Because these tasks are fictitious, we let $w_{i,Begin} = w_{i,End} = 0$ for each processor P_i. No task of type T_{begin} is consumed by any processor, and no file of type $e_{k,end}$ is sent between any processor pair, for each edge $e_{k,end} : T_k \rightarrow T_{end}$. This is ensured by the following equations:

$$\begin{array}{l} \forall P_i, cons(P_i, T_{begin}) = 0 \\ \forall P_i, \forall P_j \in n(P_i), \forall e_{k,end} : T_k \rightarrow T_{end}, \\ \quad \begin{cases} sent(P_i \rightarrow P_j, e_{k,end}) = 0 \\ sent(P_j \rightarrow P_i, e_{k,end}) = 0 \end{cases} \end{array} \quad (8)$$

Note that we can let $data_{k,end} = +\infty$ for each edge $e_{k,end} : T_k \rightarrow T_{end}$, but we need to add that $s(P_i \rightarrow P_j, e_{k,end}) = sent(P_i \rightarrow P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j}) = 0$ (in other words, $0 \times +\infty = 0$ in this equation).

2.3 Conservation laws

The last constraints deal with *conservation laws*: we state them formally, then we work out an example to help understand these constraints.

Consider a given processor P_i, and a given edge $e_{k,l}$ in the task graph. During each time unit, P_i receives from its neighbors a given number of files of type $e_{k,l}$: P_i receives exactly $\sum_{P_j \in n(P_i)} sent(P_j \rightarrow P_i, e_{k,l})$ such files. Processor P_i itself executes some tasks T_k, namely $cons(P_i, T_k)$ tasks T_k, thereby generating as many new files of type $e_{k,l}$.

What does happen to these files? Some are sent to the neighbors of P_i, and some are consumed by P_i to execute tasks of type T_l. We derive the equation:

$$\begin{array}{l} \forall P_i, \forall e_{k,l} \in E : T_k \rightarrow T_l, \\ \quad \displaystyle\sum_{P_j \in n(P_i)} sent(P_j \rightarrow P_i, e_{k,l}) + cons(P_i, T_k) = \\ \quad\quad \displaystyle\sum_{P_j \in n(P_i)} sent(P_i \rightarrow P_j, e_{k,l}) + cons(P_i, T_l) \end{array} \quad (9)$$

It is important to understand that equation (9) really applies to the steady-state operation. At the beginning of the operation of the platform, only input tasks are available to be forwarded. Then some computations take place, and tasks of other types are generated. At the end of this initialization phase, we enter the steady-state: during each time-period in steady-state, each processor can simultaneously perform some computations, and send/receive some other tasks. This is why equation (9) is sufficient, we do not have to detail which operation is performed at which time-step.

In fact, equation (9) does not hold for the master processor P_{master}, because we assume that it holds an infinite number of tasks of type T_{begin}. It must be replaced by the following equation:

$$\begin{array}{l} \forall e_{k,l} \in E : T_k \rightarrow T_l \text{ with } k \neq begin, \\ \displaystyle\sum_{P_j \in n(P_{\text{master}})} sent(P_j \rightarrow P_{\text{master}}, e_{k,l}) + cons(P_{\text{master}}, T_k) = \\ \displaystyle\sum_{P_j \in n(P_{\text{master}})} sent(P_{\text{master}} \rightarrow P_j, e_{k,l}) + cons(P_{\text{master}}, T_l) \end{array} \quad (10)$$

Note that dealing with several masters would be straightforward, by writing equation 10 for each of them.

3 Computing the optimal steady-state

The equations listed in the previous section constitute a linear programming problem, whose objective function is the total throughput, i.e. the number of tasks T_{end} consumed within one time-unit:

$$\sum_{i=1}^{p} cons(P_i, T_{end}) \quad (11)$$

Here is a summary of the linear program:

STEADY-STATE SCHEDULING PROBLEM SSSP(G)

Maximize

$TP = \sum_{i=1}^{p} cons(P_i, T_{end})$,

subject to

$\forall i, \forall k, \quad 0 \leq \alpha(P_i, T_k) \leq 1$

$\forall i, j, \forall e_{k,l} \in E, \quad 0 \leq s(P_i \rightarrow P_j, e_{k,l}) \leq 1$

$\forall i, j, \forall e_{k,l} \in E, \quad s(P_i \rightarrow P_j, e_{k,l}) = sent(P_i \rightarrow P_j, e_{k,l}) \times (data_{k,l} \times c_{i,j})$

$\forall i, \forall k, \quad \alpha(P_i, T_k) = cons(P_i, T_k) \times w_{i,k}$

$\forall i, \quad \sum_{P_j \in n(P_i)} \sum_{e_{k,l} \in E} s(P_i \rightarrow P_j, e_{k,l}) \leq 1$

$\forall i, \quad \sum_{P_j \in n(P_i)} \sum_{e_{k,l} \in E} s(P_j \rightarrow P_i, e_{k,l}) \leq 1$

$\forall i, \quad \sum_{T_k \in V} \alpha(P_i, T_k) \leq 1$

$\forall i, \quad cons(P_i, T_{begin}) = 0$

$\forall i, j, \forall e_{k,end} \quad sent(P_i \rightarrow P_j, e_{k,end}) = 0$

$\forall i, \forall e_{k,l} \in E,$

$\sum_{P_j \in n(P_i)} sent(P_j \rightarrow P_i, e_{k,l}) + cons(P_i, T_k) = \sum_{P_j \in n(P_i)} sent(P_i \rightarrow P_j, e_{k,l}) + cons(P_i, T_l)$

$\forall e_{k,l} \in E$ with $k \neq begin$,

$\sum_{P_j \in n(P_{\text{master}})} sent(P_j \rightarrow P_{\text{master}}, e_{k,l}) + cons(P_{\text{master}}, T_k) = \sum_{P_j \in n(P_{\text{master}})} sent(P_{\text{master}} \rightarrow P_j, e_{k,l}) + cons(P_{\text{master}}, T_l)$

We can state the main result of this paper:

Theorem 1. *The solution to the previous linear programming problem provides the optimal solution to* SSSP(G)

Because we have a linear programming problem in rational numbers, we obtain rational values for all variables

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

in polynomial time (polynomial in the sum of the sizes of the task graph and of the platform graph). When we have the optimal solution, we take the least common multiple of the denominators, and thus we derive an integer period for the steady-state operation.

4 Working out an example

In this section we fully work out a numerical instance of the application/architecture platform given in Figures 1 and 2. We start by extending the task graph with T_{begin} and T_{end}, as illustrated in Figure 3.

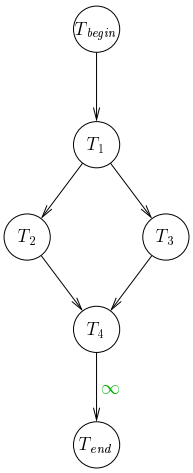

Figure 3. Extending the fork (task) graph with T_{begin} and T_{end}.

We assume that P_1 is the master processor. We outline the conservation equations which hold for P_1:

$$sent(P_2 \to P_1, e_{1,2}) + sent(P_3 \to P_1, e_{1,2}) + cons(P_1, T_1) = cons(P_1, T_2) + sent(P_1 \to P_2, e_{1,2}) + sent(P_1 \to P_3, e_{1,2})$$

$$sent(P_2 \to P_1, e_{1,3}) + sent(P_3 \to P_1, e_{1,3}) + cons(P_1, T_1) = cons(P_1, T_3) + sent(P_1 \to P_2, e_{1,3}) + sent(P_1 \to P_3, e_{1,3})$$

$$sent(P_2 \to P_1, e_{2,4}) + sent(P_3 \to P_1, e_{2,4}) + cons(P_1, T_2) = cons(P_1, T_4) + sent(P_1 \to P_2, e_{2,4}) + sent(P_1 \to P_3, e_{2,4})$$

$$sent(P_2 \to P_1, e_{3,4}) + sent(P_3 \to P_1, e_{3,4}) + cons(P_1, T_3) = cons(P_1, T_4) + sent(P_1 \to P_2, e_{3,4}) + sent(P_1 \to P_3, e_{3,4})$$

$$sent(P_2 \to P_1, e_{4,end}) + sent(P_3 \to P_1, e_{4,end}) + cons(P_1, T_4) = cons(P_1, T_{end}) + sent(P_1 \to P_2, e_{4,end}) + sent(P_1 \to P_3, e_{4,end})$$

Similarly, the following conservation equations hold for P_2:

$$sent(P_1 \to P_2, e_{begin,1}) + sent(P_3 \to P_2, e_{begin,1}) + sent(P_4 \to P_2, e_{begin,1}) + cons(P_2, T_{begin}) = cons(P_2, T_1) + sent(P_2 \to P_1, e_{begin,1}) + sent(P_2 \to P_3, e_{begin,1}) + sent(P_2 \to P_4, e_{begin,1})$$

$$sent(P_1 \to P_2, e_{1,2}) + sent(P_3 \to P_2, e_{1,2}) + sent(P_4 \to P_2, e_{1,2}) + cons(P_2, T_1) = cons(P_2, T_2) + sent(P_2 \to P_1, e_{1,2}) + sent(P_2 \to P_3, e_{1,2}) + sent(P_2 \to P_4, e_{1,2})$$

$$sent(P_1 \to P_2, e_{1,3}) + sent(P_3 \to P_2, e_{1,3}) + sent(P_4 \to P_2, e_{1,3}) + cons(P_2, T_1) = cons(P_2, T_3) + sent(P_2 \to P_1, e_{1,3}) + sent(P_2 \to P_3, e_{1,3}) + sent(P_2 \to P_4, e_{1,3})$$

$$sent(P_1 \to P_2, e_{2,4}) + sent(P_3 \to P_2, e_{2,4}) + sent(P_4 \to P_2, e_{2,4}) + cons(P_2, T_2) = cons(P_2, T_4) + sent(P_2 \to P_1, e_{2,4}) + sent(P_2 \to P_3, e_{2,4}) + sent(P_2 \to P_4, e_{2,4})$$

$$sent(P_1 \to P_2, e_{3,4}) + sent(P_3 \to P_2, e_{3,4}) + sent(P_4 \to P_2, e_{3,4}) + cons(P_2, T_3) = cons(P_2, T_4) + sent(P_2 \to P_1, e_{3,4}) + sent(P_2 \to P_3, e_{3,4}) + sent(P_2 \to P_4, e_{3,4})$$

$$sent(P_1 \to P_2, e_{4,end}) + sent(P_3 \to P_2, e_{4,end}) + sent(P_4 \to P_2, e_{4,end}) + cons(P_2, T_4) = cons(P_2, T_{end}) + sent(P_2 \to P_1, e_{4,end}) + sent(P_2 \to P_3, e_{4,end}) + sent(P_2 \to P_4, e_{4,end})$$

Newt we add numerical values for the $w_{i,k}$, the $c_{i,j}$ and the $data_{k,l}$: see Figure 4. The values of the $data_{k,l}$ are indicated along the edges of the task graph. The values of the $c_{i,j}$ are indicated along the edges of the platform graph. For the sake of simplicity, we let $w_{i,k} = w_i \times \delta_k$ for all tasks T_k, where the corresponding values for w_i are indicated close to

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

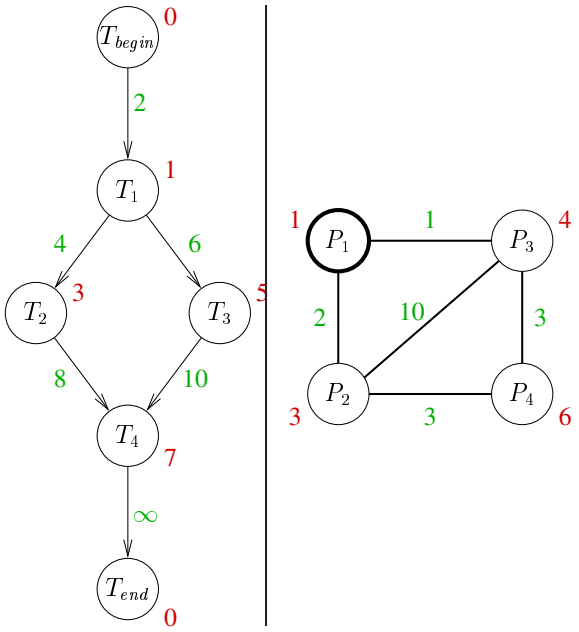

Figure 4. The application/architecture example with numerical values.

the nodes of the platform graph and the corresponding values for δ_k are indicated close to the nodes of the dependency graph. The master processor P_1 is circled in bold.

We feed the values $c_{i,j}$, $w_{i,k}$ and $data_{k,l}$ into the linear program, and compute the solution using a tool like the Maple simplex package [9]. We obtain the optimal throughput $TP = 7/64$. This means that the whole platform is equivalent to a single processor capable of processing 7 tasks every 64 seconds. The actual period is equal to 91840. The resulting values for $\alpha(P_i, T_k)$ are gathered in Table 1, and those for $cons(P_i, T_k)$ are gathered in Table 2.

	T_1	T_2	T_3	T_4
P_1	$7/64$	$13719/91840$	$7213/18368$	$4573/13120$
P_2	0	$1851/11480$	$531/1148$	$617/1640$
P_3	0	$33/82$	0	$49/82$
P_4	0	$6/41$	0	$35/41$

Table 1. Optimal solutions for $\alpha(P_i, T_k)$

	T_{begin}	T_1	T_2	T_3	T_4	T_{end}
P_1	0	$7/64$	$4573/91840$	$7213/91840$	$4573/91840$	$4573/91840$
P_2	0	0	$617/34440$	$177/5740$	$617/34440$	$617/34440$
P_3	0	0	$11/328$	0	$7/328$	$7/328$
P_4	0	0	$1/123$	0	$5/246$	$5/246$
Total						$7/64$

Table 2. Optimal solutions for $cons(P_i, T_k)$

The resulting values for $sent(P_i \rightarrow P_j, e_{k,l})$ can be summarized in the following way:

- $P_1 \rightarrow P_2$: a fraction $299/1435$ of the time is spent communicating (files for edge) $e_{1,2}$ and a fraction $531/1435$ of the time is spent communicating (files for edge) $e_{1,3}$;
- $P_1 \rightarrow P_3$: $11/82$ of the time is spent communicating $e_{1,3}$ and $165/574$ of the time is spent communicating $e_{3,4}$;
- $P_2 \rightarrow P_4$: $4/41$ of the time is spent communicating $e_{1,2}$ and $445/1148$ of the time is spent communicating $e_{3,4}$;
- $P_3 \rightarrow P_4$: $12/41$ of the time is spent communicating $e_{2,4}$ and $255/1148$ of the time is spent communicating $e_{3,4}$;
- the link $P_2 \leftrightarrow P_3$ is not used.

The relatively *weak* throughput of our platform is to be compared to the efficiency of the master processor P_1, which is the fastest one. If P_1 had been used alone, it would have been possible to process 4 tasks every 64 units of time. Instead, we achieve 7 tasks every 64 units of time, despite all the dependences, despite the communication overheads, and despite the fact that the other processors are at least three times slower than P_1. This clearly demonstrates the usefulness of deploying the target problem suite on the heterogeneous platform.

Also, it is worth pointing out that the solution is not trivial, in that processors do not execute tasks of all types. In the example, the processors are equally efficient on all task types: $w_{i,k} = w_i \times \delta_k$, hence only relative speeds count. We could have expected each problem to be processed by a single processor, that would execute all the tasks of the problem, in order to avoid extra communications; in this scenario, the only communications would correspond to the input cost $c_{begin,1} = 2$. However, the intuition is misleading. In the optimal steady state solution, some processors do not process some task types at all (see P_3 and P_4), and some task types are executed by one processor only (see T_1). This example demonstrates that in the optimal solution, the processing of each problem may well be distributed over the whole platform. This illustrates the full potential of the mixed data and task parallelism approach.

5 Related problems

We classify several related papers along the following main lines:

Scheduling task graphs on heterogeneous platforms

Several heuristics have been introduced to schedule

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

(acyclic) task graphs on different-speed processors, see Maheswaran and Siegel [18], Oh and Ha [19], Topcuoglu, Hariri and Wu [33], and Sih and Lee [27] among others. Unfortunately, all these heuristics assume no restriction on the communication resources, which renders them somewhat unrealistic to model real-life applications. Recent papers by Hollermann, Hsu, Lopez and Vertanen [14], Hsu, Lee, Lopez and Royce [15], and Sinnen and Sousa [29, 28], suggest to take communication contention into account. Among these extensions, scheduling heuristics under the one-port model (see Johnsson and Ho [16] and Krumme, Cybenko and Venkataraman [17]) are considered in [3]: just as in this paper, each processor can communicate with at most another processor at a given time-step.

Master-slave on the computational grid Master-slave scheduling on the grid can be based on a network-flow approach (see Shao, Berman and Wolski [26] and Shao [25]), or on an adaptive strategy (see Heymann, Senar, Luque and Livny [13]). Note that the network-flow approach of [26, 25] is possible only when using a full multiple-port model, where the number of simultaneous communications for a given node is not bounded. Enabling frameworks to facilitate the implementation of master-slave tasking are described in Goux, Kulkarni, Linderoth and Yoder [12], and in Weissman [34].

Mixed task and data parallelism There are a very large number of papers dealing with mixed task and data parallelism. We quote the work of Subhlok, Stichnoth, O'Hallaron and Gross [30], Chakrabarti, Demmel and Yelick [8], Ramaswamy, Sapatnekar and Banerjee [21], Bal and M. Haines [1], and Subhlok and Vondran [31], but this list is by no means meant to be comprehensive. We point out, however, that (to the best of our knowledge) none of the papers published in this area is dealing with heterogeneous platforms. In fact, Taura and Chien [32] do consider the pipeline execution of task graphs onto heterogeneous platforms, but they make the restrictive hypothesis that all copies of a given task type must be executed on the same processor.

Asymptotic results Bertsimas and Gamarnik [6] have used a fluid relaxation technique (inspired by the work of Sevast'janov [23, 24]) to derive asymptotically optimal scheduling algorithms. They apply this technique to the job shop scheduling problem and to the packet routing problem. It would be very interesting to extend these results to a heterogeneous framework.

6 Conclusion

In this paper, we have dealt with the implementation of mixed task and data parallelism onto heterogeneous platforms. We have shown how to determine the best steady-state scheduling strategy for a general task graph and for a general platform graph, using a linear programming approach.

This work can be extended in the following two directions:

- On the theoretical side, we could try to solve the problem of maximizing the number of tasks that can be executed within K time-steps, where K is a given time-bound. This scheduling problem is more complicated than the search for the best steady-state. Taking the initialization phase into account renders the problem quite challenging.
- On the practical side, we need to run actual experiments rather than simulations. Indeed, it would be interesting to capture actual architecture and application parameters, and to compare heuristics on a real-life problem suite.

References

[1] H. Bal and M. Haines. Approaches for integrating task and data parallelism. *IEEE Concurrency*, 6(3):74–84, 1998.

[2] C. Banino, O. Beaumont, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor grids. In *PARA'02: International Conference on Applied Parallel Computing*, LNCS. Springer Verlag, 2002.

[3] O. Beaumont, V. Boudet, and Y. Robert. A realistic model and an efficient heuristic for scheduling with heterogeneous processors. In *HCW'2002, the 11th Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2002.

[4] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium IPDPS'2002*. IEEE Computer Society Press, 2002. Extended version available as LIP Research Report 2001-25.

[5] O. Beaumont, A. Legrand, and Y. Robert. The master-slave paradigm with heterogeneous processors. In D. S. Katz, T. Sterling, M. Baker, L. Bergman, M. Paprzycki, and R. Buyya, editors, *Cluster'2001*, pages 419–426. IEEE Computer Society Press, 2001. Extended version available as LIP Research Report 2001-13.

[6] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithm for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[7] T. D. Braun, H. J. Siegel, and N. Beck. Optimal use of mixed task and data parallelism for pipelined computations. *J. Parallel and Distributed Computing*, 61:810–837, 2001.

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 IEEE

[8] S. Chakrabarti, J. Demmel, and K. Yelick. Models and scheduling algorithms for mixed data and task parallel programs. *J. Parallel and Distributed Computing*, 47:168–184, 1997.

[9] B. W. Char, K. O. Geddes, G. H. Gonnet, M. B. Monagan, and S. M. Watt. *Maple Reference Manual*, 1988.

[10] J. Cowie, B. Dodson, R.-M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, and J. Zayer. A world wide number field sieve factoring record: on to 512 bits. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology - Asiacrypt '96*, volume 1163 of *LNCS*, pages 382–394. Springer Verlag, 1996.

[11] Entropia. URL: `http://www.entropia.com`.

[12] J. P. Goux, S. Kulkarni, J. Linderoth, and M. Yoder. An enabling framework for master-worker applications on the computational grid. In *Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*. IEEE Computer Society Press, 2000.

[13] E. Heymann, M. A. Senar, E. Luque, and M. Livny. Adaptive scheduling for master-worker applications on the computational grid. In R. Buyya and M. Baker, editors, *Grid Computing - GRID 2000*, pages 214–227. Springer-Verlag LNCS 1971, 2000.

[14] L. Hollermann, T. S. Hsu, D. R. Lopez, and K. Vertanen. Scheduling problems in a practical allocation model. *J. Combinatorial Optimization*, 1(2):129–149, 1997.

[15] T. S. Hsu, J. C. Lee, D. R. Lopez, and W. A. Royce. Task allocation on a network of processors. *IEEE Trans. Computers*, 49(12):1339–1353, 2000.

[16] S. L. Johnsson and C.-T. Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, 1989.

[17] D. W. Krumme, G. Cybenko, and K. N. Venkataraman. Gossiping in minimal time. *SIAM J. Computing*, 21:111–139, 1992.

[18] M. Maheswaran and H. J. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems. In *Seventh Heterogeneous Computing Workshop*. IEEE Computer Society Press, 1998.

[19] H. Oh and S. Ha. A static scheduling heuristic for heterogeneous processors. In *Proceedings of Europar'96*, volume 1123 of *LNCS*, Lyon, France, Aug. 1996. Springer Verlag.

[20] Prime. URL: `http://www.mersenne.org`.

[21] S. Ramaswamy, S. Sapatnekar, and P. Banerjee. A framework for exploiting task and data parallelism on distributed memory multicomputers. *IEEE Trans. Parallel and Distributed Systems*, 8(11):1098–1116, 1997.

[22] SETI. URL: `http://setiathome.ssl.berkeley.edu`.

[23] S. V. Sevast'janov. An algorithm with an estimate for a problem with routings of parts of arbitrary shape and alternative executors. *Cybernetics*, 22:773–781, 1986.

[24] S. V. Sevast'janov. On some geometric methods in scheduling theory: a survey. *Discrete Applied Mathematics*, 55:59–82, 1994.

[25] G. Shao. *Adaptive scheduling of master/worker applications on distributed computational resources*. PhD thesis, Dept. of Computer Science, University Of California at San Diego, 2001.

[26] G. Shao, F. Berman, and R. Wolski. Master/slave computing on the grid. In *Heterogeneous Computing Workshop HCW'00*. IEEE Computer Society Press, 2000.

[27] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–187, 1993.

[28] O. Sinnen and L. Sousa. Comparison of contention-aware list scheduling heuristics for cluster computing. In T. M. Pinkston, editor, *Workshop for Scheduling and Resource Management for Cluster Computing (ICPP'01)*, pages 382–387. IEEE Computer Society Press, 2001.

[29] O. Sinnen and L. Sousa. Exploiting unused time-slots in list scheduling considering communication contention. In R. Sakellariou, J. Keane, J. Gurd, and L. Freeman, editors, *EuroPar'2001 Parallel Processing*, pages 166–170. Springer-Verlag LNCS 2150, 2001.

[30] J. Subhlok, J. Stichnoth, D. O'Hallaron, and T. Gross. Exploiting task and data parallelism on a multicomputer. In *Fourth ACM SIGPLAN Symposium on Priciples & Practices of Parallel Programming*. ACM Press, May 1993.

[31] J. Subhlok and G. Vondran. Optimal use of mixed task and data parallelism for pipelined computations. *J. Parallel and Distributed Computing*, 60:297–319, 2000.

[32] K. Taura and A. A. Chien. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *Heterogeneous Computing Workshop*, pages 102–115. IEEE Computer Society Press, 2000.

[33] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *Eighth Heterogeneous Computing Workshop*. IEEE Computer Society Press, 1999.

[34] J. B. Weissman. Scheduling multi-component applications in heterogeneous wide-area networks. In *Heterogeneous Computing Workshop HCW'00*. IEEE Computer Society Press, 2000.

Proceedings of the Eleventh Euromicro Conference on Parallel,Distributed and Network-Based Processing (Euro-PDP'03)
0-7695-1875-3/03 $17.00 © 2003 **IEEE**