TASK REDISTRIBUTION SCHEDULING USING MULTI-MASTER DIVISIBLE LOAD MODEL

Reiji Suda  Sayaka Temi
Department of Computer Science  Department of Computer Science
the University of Tokyo  the University of Tokyo
7-3-1 Hongo, Bunkyo-ku  7-3-1 Hongo, Bunkyo-ku
113-0033 Tokyo, Japan  113-0033 Tokyo, Japan
(email: reiji@is.s.u-tokyo.ac.jp  (Currently with INCS INC.)

ABSTRACT

This paper proposes a scheduling algorithm for redistribution of uniform independent divisible tasks on heterogeneous processors connected by a homogeneous network. The proposed algorithm is based on the multi-master divisible load model, which is an extension of the divisible load model from the master-worker framework. The schedule is asymptotically optimum, that is, the ratio of the makespan against the optimum one approaches to the unity when the amount of tasks grows infinitely. The computational complexity is so low that it can be used for run-time redistribution and/or rescheduling. In experiments it gives makespans very close to the optimums, and it is more efficient compared to some existing methods.

KEY WORDS
Task Scheduling, Algorithm for Heterogeneous Systems, Load balancing, Communication Algorithm, Real-time scheduling, Divisible Load Theory

1 Introduction

Redistribution of tasks or data is one of the most fundamental problems of parallel processing. In works of data redistribution on the data parallel paradigm, the communication phase and the computation phase are usually separated, and the communication time is minimized under the restriction that the computational costs are equal on all processors, but that is not optimum especially with heterogeneous processors. MinEX [1] is a rare partitioner which takes redistribution costs into account, although it does not schedule communications.

Job scheduling algorithms [2] schedule both communications and computations, but precise optimization is computationally intractable in most cases. Several approximate algorithms for the task scheduling problems are known, but the authors know none for task redistribution problem with communication delays. There are many works on redistribution scheduling in master-worker model (for example [3] as one of the best results), most of them are found in the literature of the Divisible Load Theory (DLT). DLT has been researched for about thirty years, and most of the works assume the master-worker model [4]. Since we have no room to review its vast literature here, we only refer a survey [5] and an exhaustive list [6].

This paper proposes an asymptotically optimum redistribution scheduling of a divisible task among heterogeneous processors. It extends DLT to a generalized model with multiple sources of tasks, which we call Multi-Master Divisible Load (MMDL) model. We will present an asymptotically optimum scheduling algorithm with a method to optimize the number and the sizes of the rounds (the definition of rounds are given in Sect. 3). Also some experimental results that confirm the usefulness of our algorithm will be given.

Among works on DLT, Hakkad [7, 8] considered a model very similar to ours. He solved only a part of the problem, that is, the cases where no processor needs idle time waiting for coming task. Our algorithm gives the optimum schedule for such a case, and otherwise an asymptotically optimum schedule. Banino et al. [9] includes discussion about DLT with multiple sources, although that is mostly theoretical, giving asymptotically optimum scheduling via linear program, which may take long time for large problems. Our algorithm solves restricted cases of theirs, but with complexity only \(O(p \log p)\) with \(p\) being the number of processors, and thus our algorithm will be applicable to run-time redistribution. Also we provide optimization of the number and the sizes of the rounds.

Yang et al. [10] argues the "optimum" number and sizes of the rounds in the master-worker-type DLT. They allow idle times only in the first round, but our algorithm removes that restriction, thus possibly gives a better solution in some cases. Also our algorithm is a non-trivial extension to the multi-master model.

2 Problem Statement

This section defines our problem, for which a solution is given in the next section. The model allows some extensions (see Sect. 5), but only the simplest model is considered in this paper for brevity.

The set of tasks can be divided into chunks of arbitrary sizes (divisible load). The costs for processing and transferring a chunk of tasks (just a task hereafter) are sim-
ple functions of the size of task. There is no dependency or need of communication between tasks, and any processor can process any task.

There are \( p \) processors, and task of size \( x_i \) is initially allocated to the \( i \)th processor. We removed the master-worker assumption, i.e., \( x_i = 0 \) for \( i \neq 0 \). The time to process task of size \( x \) on the \( i \)th processor is \( \gamma_i x \) (heterogeneous processors).

The processors are connected by a homogeneous network, on which the time to transfer task of size \( x \) between any pair of processors is \( \alpha + \beta x \). It is independent of concurrent communications if the sender-receiver pairs are disjoint (crossbar network and single port). A processor cannot process and transfer tasks at the same time (no overlap of communication and computation).

The problem is to schedule computations and communications with the makespan (or schedule length) as short as possible. All the tasks must be processed, but the results are not to be collected.

3 Redistribution Scheduling Algorithm

3.1 Restrictions on the Schedules

We introduce the following restrictions on the schedules.

(a) Although the communication time is modeled as \( \alpha + \beta x \), the constant term \( \alpha \) is ignored in most parts of the scheduling algorithm.

(b) The schedule consists of (one or more) rounds, which is a partial schedule for some time interval. The rounds are similar (with equal or different sizes) to each other, with possible exceptions at the first and the last rounds, which are called prologue and epilogue, respectively (see Figure 1). The other rounds are called regular rounds.

(c) There is no relay, that is, no processor does both send and receive tasks.

Restriction (c) classifies the processors into masters that send some task to other processors, workers that receive some task from other processors, and independents that do not communicate with other processors.

The sizes of the rounds are assumed to be the same (uniform-sized rounds) in the scheduling algorithm in Sect. 3.2, and then in Sect. 3.3 they become variable (variable-sized rounds) and are optimized.

Asymptotic Optimality of the Schedule. Now we prove those restrictions allow asymptotically optimum schedule with uniform-sized rounds. The proof is almost parallel to that of Beaumont et al. [11].

The scheduling with uniform-sized rounds is defined as follows (Figure 1). The regular rounds have the same sizes, and thus they are identical. The task processed by a worker in a round is transferred from masters in the previous round. This rule is also applied to the prologue and the epilogue, therefore the prologue is defined by removing computations from a regular round, and the epilogue is defined by removing communications from a regular round. The schedule of a regular round is optimized for \( \alpha = 0 \), discussed in Sect. 3.2.

![Figure 1. Scheduling with uniform-sized rounds](image)

**Theorem 1.** The above schedule is asymptotically optimum, that is, as the total size of the task increases infinitely, the ratio of the makespan of the obtained schedule and that of the optimum one approaches to the unity.

**Proof:** Let the number of the rounds, which includes the prologue but not the epilogue, be \( k \). Thus each regular round processes \( 1/k \) of the task. Let the time taken to the regular rounds with \( \alpha = 0 \) be \( T/k \). Since the times for the prologue and the epilogue are not longer than \( T/k \), the makespan must not be larger than \( T + T/k \). Then the makespan approaches to \( T \) for \( k \rightarrow \infty \), which is optimum for \( \alpha = 0 \) and is a lower bound of the makespan for \( \alpha > 0 \).

Let the schedule length of a regular round with \( \alpha > 0 \) be \( T/k + A \) with a constant \( A \) independent of \( T \) and \( k \) (actually \( A = n \alpha \) with \( n \) being the maximum number of sends or receives of a processor per round). New the makespan is bounded by \( T + T/k + A k \), which is \( T + 2 \sqrt{A T} \) for \( k = \sqrt{T/A} \). Since the optimum makespan cannot be less than \( T \), the proposed schedule is no longer than \( 1 + 2 \sqrt{A T} \) times the optimum schedule. \( A \) being a constant, the factor \( 1 + 2 \sqrt{A T} \) approaches to the unity for large \( T \).

We will prove constructively that the restriction (c) does not prevent optimum schedule of regular rounds for \( \alpha = 0 \).

3.2 Optimum Schedule with Uniform-Sized Rounds for \( \alpha = 0 \)

This subsection gives an algorithm of optimum scheduling with uniform-sized regular rounds for \( \alpha = 0 \). Let \( k = 1 \), since \( \alpha = 0 \) makes the computation/communication times linear and thus optimum schedule independent of \( k \).
Determining Optimum $T$. Define $y_i$ as follows: If $y_i > 0$ then the $i$th processor is a worker and receives task of size $y_i$; if $y_i < 0$ then the $i$th processor is a master and sends task of size $-y_i$; and if $y_i = 0$ then the $i$th processor is an independent.

Because every processor must complete the computations and the communications in time $T$, we have $(x_i + y_i) \gamma_i + \beta |y_i| \leq T$. This is equivalent to

$$y_i \leq \bar{y}_i := \begin{cases} (T - x_i \gamma_i) / (\gamma_i - \beta) & \text{if } x_i \gamma_i > T \\ (T - x_i \gamma_i) / (\gamma_i + \beta) & \text{otherwise} \end{cases} \quad (1)$$

Since $x_i + y_i$ is the size of task processed by the $i$th processor, it must be non-negative. This requires

$$T \geq T_L := \max_i \min \{x_i \beta, x_i \gamma_i\} \quad (2)$$

This guarantees that the denominators in (1) are positive. The total amount of the sent tasks must equal the total amount of the received tasks:

$$\sum_i y_i = 0 \quad (3)$$

If $\sum_i \bar{y}_i \geq 0$ holds for $T \geq T_L$, then we have some $y_i$ that satisfies (1) to (3). Because every $\bar{y}_i$ is a continuous increasing function of $T$, if $T = T_L$ gives $\sum_i \bar{y}_i \geq 0$, then $T_L$ is the shortest possible $T$. Otherwise we have some $T > T_L$ that gives $\sum_i \bar{y}_i = 0$. Because

$$\bar{Y}(T) := \sum_i \bar{y}_i(T) \quad (4)$$

is piecewise linear with knots $\{x_i \gamma_i\}$, the equation $\bar{Y}(T) = 0$ can be solved by binary search and linear interpolation.

The algorithm to get the optimum $T$ is given in Figure 2. The computational complexity is $O(p \log p)$, coming from the sorting of step 3 and binary search of step 4, since each evaluation of $\bar{Y}(T)$ requires $O(p)$ time.

1. Compute $T_L$ defined by (2).
2. If $\bar{Y}(T_L) \geq 0$ then return $T_L$.
3. Sort processor indices so that $x_{i+1} \gamma_{i+1} \leq x_{i+1} \gamma_{i+1}$.
4. Obtain $j$ such that $\bar{Y}(x_{i+1} \gamma_{i+1}) \leq 0$ and $\bar{Y}(x_{i+1} \gamma_{i+1}) \geq 0$ by binary search.
5. Solve $Y(T) = 0$ by linear interpolation from $\bar{Y}(x_{i+1} \gamma_{i+1}) \leq 0 \leq \bar{Y}(x_{i+1} \gamma_{i+1})$.

Figure 2. Computing the optimum $T$

Master-Worker Matching. Next it is determined how much task is sent from which master to which worker. We propose the following interval intersection matching to that purpose. The simplicity of the algorithm is important in the remaining parts of the scheduling algorithm.

Let the master processors be $m_1, m_2, \ldots, m_M$ and the worker processors be $n_1, n_2, \ldots, n_N$, where $M$ and $N$ are the numbers of the masters and the workers, respectively.

New define $I_0 = 0$ and $I_i = I_{i-1} + |y_{m_i}|$. The master processor $m_i$ is associated with the interval $[I_{i-1}, I_i]$ of length $|y_{m_i}|$. Similarly the worker processor $n_j$ is associated with the interval $[J_{j-1}, J_j]$ of length $|y_{n_j}|$. By letting $J_0 = 0$ we have $I_M = J_N$ from (3).

New determine the amount of task sent from the master processor $m_i$ to the worker processor $n_j$ as

$$y_{ij} = \max \{0, \min \{I_i, J_j\} - \max \{I_{i-1}, J_{j-1}\}\} \quad (5)$$

which is the length of the intersection of the intervals $[I_{i-1}, I_i]$ and $[J_{j-1}, J_j]$.

It is easy to see that no more than $p - 1$ of $y_{ij}$ are non-zero, and they can be computed in $O(p)$ time by “merging” two lists $\{I_i\}$ and $\{J_j\}$, just as the merge sort algorithm.

Communication Scheduling. Next the communications in a regular round are scheduled. Since we assumed that the communications needed for the computations in a round is done in the previous round, it is enough to insert the computations in the time intervals without communications.

We call the following scheduling scheme reverse order scheduling of communications, which is a kind of list scheduling. The communications are scheduled as early as possible in a round. If a processor has more than one communications, then they are scheduled in the following order. If the processor is a master, then the communications for workers of smaller indices are scheduled first. If the processor is a worker, then the communications for masters of larger indices are scheduled first. Note that the priority is reverse for master and worker.

This is optimum in the following sense.

Theorem 2. Assume that the interval intersection matching is used. Then the reverse order scheduling gives a schedule that finishes the communications earliest.

Proof: If a processor communicates to more than one processors, then it does not wait but the last one. This is because the first communication is also the first one for the partner of the communication, and the other communications but the last one is the only one for the partner.

Now consider the communication that finishes last. It starts when both of the sender and the receiver finish the other communications of theirs, therefore (at least) one of them need not wait. That processor does not wait at all, and thus the communications cannot finish earlier.

Note that, from the proof, the communications of the processor with the longest communication time do not wait. Since the sum of communication times of a processor is not longer than $T$ because of (1), all communications are scheduled within time $T$. Because $T$ is a lower bound of the makespan, the scheduling algorithm in this section, summarized in Figure 3, gives an optimum schedule for uniform-sized rounds with $\alpha = 0$.  

---

**Figure 2.** Computing the optimum $T$.
1. Compute the optimum $T$ as in Figure 2.
2. Determine $y_i$ from Eqs. (1) and (3).
3. Compute $I_i$ and $J_i$.
4. Determine non-zero $y_{i,j}$ from Eq. (5).
5. Do reverse order scheduling for communications, and fill remaining time slots with computations.

Figure 3. Scheduling for uniform-sized rounds

The computational complexity for scheduling communications is $O(p)$ since there are no more than $p-1$ messages. Thus the total complexity of scheduling uniform-sized rounds is $O(p \log p)$.

### 3.3 Optimizing Number and Sizes of Rounds

Next, the schedule is improved by allowing variable sizes of the rounds (see Figure 4). Let the size of the $n$th round be $r_n$ that is, the schedule of the $n$th round is obtained by scaling the uniform regular round schedule by the factor of $r_n = 1$, $r_n$ is a fraction of the total makespan occupied by the $n$th round. Define

$$R_n = \sum_{i=0}^{j-1} r_i,$$  \hspace{1cm} (6)

where the prologue is considered as the 0th round and $R_0 = 0$. We have $R_k = 1$ in order to process all tasks, where $k$ is the number of rounds except epilogue.

![Figure 4. Schedule with variable-sized rounds](image)

Before optimizing the number and the sizes of the rounds, the schedule is normalized. First, dummy tasks are added so that

$$(x_i + y_i)\gamma_i + \beta |y_i| = T$$  \hspace{1cm} (7)

holds on every processor. Second, if the schedule has a time interval with no processor communicating, then it is moved to the epilogue (see Figure 4). For simplicity $T$ is redefined as the time excluding the time interval moved to the epilogue.

Although the idle time of the schedule is removed by the dummy tasks, we allow additional idle time $\zeta$ for every processor. This is equivalent to introduce imaginary task of size $z_i = \zeta/\gamma_i$.

**The Schedulability Condition.** Next consider the condition that the tasks arrive at the worker before they are processed.

It can be derived from the proof of Theorem 2 that the schedule of the $n$th processor in a round can be summarized as: (i) communicating task of size $w_{1,n}$, (ii) processing task of size $w_{2,n}$, (iii) communicating task of size $w_{3,n}$, and (iv) processing task of size $w_{4,n}$, where some of $w_{i,n}$ may be zero.

First consider the first computation (ii) of the prologue on the $n$th processor. The required size of task is $r_0 w_{1,n}$, and that must be available after the first communication (i), thus it must hold that

$$r_0 w_{1,n} \leq x_i + z_i + r_0 w_{1,n}.$$

For the second computation (iv) of the prologue we have

$$r_0 w_{4,n} \leq x_i + z_i + r_0 (w_{1,n} - w_{2,n} + w_{3,n}).$$

It can be shown that (9) is always satisfied, although we omit the proof.

Considering similarly, it is required that

$$r_j (w_{2,n} - w_{1,n}) \leq x_i + z_i - r_j (w_{4,n} - w_{1,n} - w_{2,n} - w_{3,n})$$

for the $n$th round, which is valid for the prologue as well. Again we omit the proof, but it can be shown that if $w_{2,n} \leq w_{1,n}$ then (8) and (10) are always satisfied. Thus we can rewrite the condition (10) as

$$r_j \leq \frac{x_i + z_i}{w_{2,n} - w_{1,n}} - R_j \frac{w_{1,n} - w_{3,n} + w_{2,n} - w_{1,n}}{w_{2,n} - w_{1,n}}$$

for workers with $w_{2,n} > w_{1,n}$. This is the schedulability condition.

**Optimum Number and Sizes of the Rounds.** Now recall $\alpha > 0$. It is better to reduce the number of rounds, or in other words, to size the rounds as large as possible. This results in

$$r_j = \min_i \left\{ \frac{x_i + z_i}{w_{2,n} - w_{1,n}} - R_j \frac{w_{1,n} - w_{3,n} + w_{2,n} - w_{1,n}}{w_{2,n} - w_{1,n}} \right\}$$

except the last round, whose size is determined by $r_{k-1} = 1 - R_{k-1}$.

If minimum of (12) is always attained by the same $i$, then the resulting round sizes are geometric, increasing or decreasing. In general, the round sizes may increase first, and then decrease.

The number and the sizes of the rounds are functions of $z_i = \zeta/\gamma_i$. A longer additional idle time $\zeta$ gives a larger round size $r_j$, a smaller number of rounds $k$, and
Schedule for uniform-sized rounds and $\zeta$ are given.
1. Let $j = 0$ and $R_0 = 0$.
2. Compute $r_j$ by Eq. (12).
3. If $R_j + r_j \geq 1$ then $r_j = 1 - R_j$ and return.
4. Let $R_{j+1} = R_j + r_j$, increment $j$ and go to step 2.

Figure 5. Determining number and sizes of rounds

less overhead of $\alpha > 0$. Here is a tradeoff, and there exists some optimum $\zeta$ that minimizes the makespan. The objective function (makespan) is not convex for $\zeta$ in the strict sense, but the golden section search gives satisfactory approximate solutions in practice. It is easy to see that the number of rounds is finite for $\zeta > 0$ and that $k = 1$ for

$$
\zeta = \max \{ \gamma_i (w_{2j} - u_{1i} - x_i) \},
$$

which can be used as the initial values for the golden section search.

Summing up the above, the proposed scheduling algorithm is shown in Figure 6. Theoretically the most time-consuming part is step 3, the golden section search, where the makespans must be evaluated repeatedly, but it will not be impractical in many cases. It can be shown that the number of rounds $k$ can be computed in time $O(p \log p)$ (we omit the proof here), and thus if it is allowed to estimate the makespan as $T_k + kA + \zeta$, then the complexity of scheduling algorithm is $O(cp \log p)$, where $c$ is the number of iterations of the golden section search. One can prove that if optimum $\zeta = \zeta_{opt}$ gives the number of rounds $k^*$, then $\zeta = \zeta_{opt} + 4$ still gives the same number of rounds $k$ (again we omit the proof). Thus $c = O(\log (\zeta_{max}/A))$ iterations are enough for golden section search.

1. Schedule uniform-sized rounds as in Figure 3.
2. Normalize schedule as discussed in Sect. 3.3.
3. Do golden section search for $\zeta$ so to minimize estimated makespan, beginning with $\zeta_{min} = 0$ and $\zeta_{max} = \max \{ \gamma_i (w_{2j} - u_{1i} - x_i) \}$.
4. Compute the sizes of rounds as in Figure 5.
5. Schedule communications from the schedule for uniform-sized rounds and $r_j$'s.
6. Fill remaining time slots with computations as much as tasks are available.
7. Schedule all un-processed tasks in the epilogue.

Figure 6. Proposed scheduling algorithm

<table>
<thead>
<tr>
<th>Proc/Load</th>
<th>Cerv.</th>
<th>Work</th>
<th>Proposed method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc</td>
<td>Load</td>
<td>exist.</td>
<td>stealing</td>
</tr>
<tr>
<td>A/1</td>
<td>5.8007</td>
<td>5.2581</td>
<td>5.2205</td>
</tr>
<tr>
<td>A/2</td>
<td>5.5001</td>
<td>5.9789</td>
<td>5.5001</td>
</tr>
<tr>
<td>A/4</td>
<td>7.2507</td>
<td>6.0750</td>
<td>5.5007</td>
</tr>
<tr>
<td>B/2</td>
<td>7.8007</td>
<td>6.8780</td>
<td>5.7125</td>
</tr>
<tr>
<td>B/3</td>
<td>6.2007</td>
<td>5.4646</td>
<td>5.3155</td>
</tr>
<tr>
<td>B/4</td>
<td>8.0007</td>
<td>6.8780</td>
<td>5.7777</td>
</tr>
<tr>
<td>C/1</td>
<td>7.8007</td>
<td>8.6589</td>
<td>5.7167</td>
</tr>
<tr>
<td>C/3</td>
<td>6.0001</td>
<td>13.2966</td>
<td>6.0001</td>
</tr>
<tr>
<td>C/4</td>
<td>5.9507</td>
<td>5.8080</td>
<td>5.1770</td>
</tr>
</tbody>
</table>

Table 1. Makespan obtained by the simulation

0.21 GHz for $i$th processor, and (case C) 2.5 GHz for $i \leq 3$ and 0.5 GHz for $i \geq 4$.

The initial task distributions are: (case D) 40,000 + 10,000 for $i$th processor, (case 2) 125,000 for $i \leq 3$ and 25,000 for $i \geq 4$, (case 3) 25,000 for $i \leq 3$ and 125,000 for $i \geq 4$, and (case 4) 120,000 for $i \leq 4$ and 0 for $i \geq 5$.

Other parameters are $\alpha = 100 \mu s, 1/b = 800 \text{ Mbps}$, computation of unit task = 100 k clocks, and communication of unit task = 1 k bytes.

The resulting makespans are given in Table 1. The processor speeds and the initial loads are shown in the column “Proc/Load.” “Cerv. exist.” shows the makespans by the conventional redistribution scheme in which the loads after redistribution are proportional to the processor speed. “Work stealing” gives the makespans by the work-stealing, a famous dynamic load balancing scheme, where the block size is optimized for each instance, and overheads, such as gathering load information and determining the processor to request tasks, are assumed to be zero. “Proposed method” presents the makespans by our algorithm, both for the schemes of uniform and optimum sizes of rounds. “Lower bound” tabulates $T_s$, which is a lower bound of the makespan.

The proposed method gives constantly good schedules, especially with the optimized round sizes. The makespans of our scheme are very close to the lower bounds. We have no room to show more simulation results with different parameters, but those conventional methods hardly defeat our algorithms.

Execution on Actual Machines. We setup a heterogeneous cluster with Xeon processors of clocks of 2.0, 2.4, 3.0 and 3.8GHz, connected by a Giga-bit ether network, and implemented our algorithm on MPJ (LAM/7C) using C language. The unit task is a multiplication of a fixed matrix and a vector. Every machine knows the fixed matrix, and vectors are transferred as tasks.

Again the number of processors is eight, and the processor speeds are: (case D) 2.4 GHz for all, (case E) two 2.0 GHz processors, four 2.4 GHz processors, and one pro-

4 Experiments

This section evaluates the proposed algorithm with simulations and with executions on actual machines.

Simulations. The first evaluation is by simulations. The models of the processors and the network are as assumed in the previous sections.

The number of processor is eight, and the processor speeds are: (case A) 1.5 GHz for all, (case B) 0.8 +
processors for each of 3.0 GHz and 3.8 GHz, and (case F) one 2.0 GHz processor, six 2.4 GHz processors, and one 3.8 GHz processor.

The initial task distributions are: (case 5) 7,500 for all, (case 6) 11,250 for first four processors and 3,750 for the others, and (case 6) 25,000 for two processors, zero for four processors, and 5,400 for the last two processors.

The performance parameters have been obtained from small-scaled trial executions. The low complexity of our algorithm makes (re)scheduling after performance evaluation via trial execution practical. The effectual clock rates were evaluated as 1.32, 1.60, 2.68, and 3.40 GHz, for 2.0, 2.4, 3.0 and 3.8 GHz machines, respectively. Communication performance parameters were estimated as \( \alpha = 18 \, \mu s \) and \( 1/\beta = 72.15 \, MB/s \).

The schedule lengths (as Sched. len) and the actual execution times (as Ex. time), accompanied by the relative error of the schedule lengths against the actual execution times (as Rel. error), are shown in Table 2, where matrix size is \( 100 \times 500 \). The execution time does not include times for trial executions and scheduling. The schedule lengths are very close to the execution times, and the difference is mostly less than one percent. We have several dozens of experiments with different initial loads and different matrix sizes, and the relative error of 3.2 percent for the case E/6 was the worst one.

<table>
<thead>
<tr>
<th>Proc/Load</th>
<th>Sched. len.</th>
<th>Ex. time</th>
<th>Rel. error</th>
</tr>
</thead>
<tbody>
<tr>
<td>D/5</td>
<td>5.8392</td>
<td>5.8526</td>
<td>0.00228</td>
</tr>
<tr>
<td>D/6</td>
<td>6.0607</td>
<td>6.0822</td>
<td>0.00353</td>
</tr>
<tr>
<td>D/7</td>
<td>6.4180</td>
<td>6.4145</td>
<td>0.00054</td>
</tr>
<tr>
<td>E/5</td>
<td>5.1033</td>
<td>5.1762</td>
<td>0.01466</td>
</tr>
<tr>
<td>E/6</td>
<td>5.2621</td>
<td>5.4358</td>
<td>0.03195</td>
</tr>
<tr>
<td>E/7</td>
<td>5.7117</td>
<td>5.8969</td>
<td>0.02233</td>
</tr>
<tr>
<td>F/5</td>
<td>5.3383</td>
<td>5.4563</td>
<td>0.02162</td>
</tr>
<tr>
<td>F/6</td>
<td>5.5273</td>
<td>5.5729</td>
<td>0.00836</td>
</tr>
<tr>
<td>F/7</td>
<td>5.7433</td>
<td>5.8424</td>
<td>0.01696</td>
</tr>
</tbody>
</table>

5 Conclusion

In this paper, we introduce the Multi-Master Divisible Load (MMDL) model for scheduling of task redistribution. This paper discusses an asymptotically optimum scheduling algorithm for the simplest problem with practical computational complexity. The resulting schedules are much better than some existing schemes.

The assumptions introduced in Sect. 2 can be extended in some ways. We have obtained asymptotically optimum scheduling algorithms with low complexity for problems with (1) overhead of message buffering (independent of the partner of communication), (2) concurrent computations and communications (with the same communication performance under computations), (3) collecting results of tasks into the originating processors, and (4) on cluster of clusters (with possibly different network speeds). Those results will be presented on other occasions.

Acknowledgements

This research was partly supported by the 21st Century COE program and Grants-in-Aid for Scientific Research (MEXT JAFAN).

References


