Divisible Load Scheduling on a Hypercube Cluster with Finite-size Buffers and Granularity Constraints

Xiaolin Li, MIEEE, Bharadwaj Veeravalli, MIEEE and Chi Chung Ko, Sr MIEEE
Open Source Software Laboratory
Department of Electrical & Computer Engineering
The National University of Singapore
10 Kent Ridge Crescent, Singapore, 119260
Email: {engp8584, elebv, elekocc@nus.edu.sg}

Abstract

In this paper, we address the problem of scheduling a large size divisible load on a hypercube cluster of processors. Unlike in earlier studies in the divisible load theory (DLT) literature, here, we assume that the processors have finite-size buffers. Further, we impose constraints on the extent to which the load can be divided, referred to as granularity constraint. We first present the closed-form solutions for the case with infinite-size buffers. For the case with load granularity constraint, we propose a simple algorithm to find the sub-optimal solution and then we analyze the case when these buffer are of finite size. For this case, we present an elegant strategy, referred to as Incremental Balancing Strategy *(IBS), to obtain an optimal load distribution. Based on the rigorous mathematical analysis, a number of interesting and useful properties exhibited by the algorithm are proven. Numerical examples are presented for the ease of understanding.*

Keywords: *Divisible load theory, hypercube, finite-size buffer, communication delays, granularity*

1. Introduction

One of the primary issues in the area of parallel and distributed computing is on how to partition and schedule loads among the available system resources so that the best performance is achieved in terms of finish time of the input tasks. The divisible load scheduling in multiprocessor systems, commonly referred to as *divisible load theory* (DLT) in the literature, is of recent origin [4]. As a new paradigm, DLT offers the ease of analytical tractability and produces richer set of results in the load scheduling in heterogeneous computing systems.

In most of the work conducted so far in DLT, the available resources at the processing site are always assumed to be plentiful. For instance, the available buffer size to hold the divisible data to be processed at a site is assumed to be infinite [2]. However, in practice, each site has a finite buffer capacity to hold the load. Thus, we are naturally confronted with a problem of scheduling a divisible load under the finite-size buffer constraints. In this paper, we consider the above-mentioned scenario on a processor clusters assuming an m-dimensional hypercube topology in which each of the processors has finite-size buffers. Also, we impose the constraint on the extent to which the divisible load can be partitioned, as in any realistic situation, a task cannot be infinitely divided. This is referred to as the granularity constraint. The problem of scheduling with finite-size buffers in a heterogeneous single-level tree network was investigated in [6]. Relevant applications of this kind of scheduling problems can be found in the literature [3].

This paper is organized as follows. We first present the problem formulation in Section 2. In Section 3, we derive the optimal solutions when there is no buffer size limitation and propose a simple algorithm to tackle the scheduling problem including the granularity constraint. In Section 4, we propose an efficient algorithm IBS (Incremental Balancing Strategy) and prove some important results. In Section 5, we present the discussions and conclusions.

2. Problem Setting and Some Preliminary Remarks

We consider a homogeneous hypercube network for analysis. A four-dimensional hypercube is shown in Figure 1. In the hypercube, each processing element (PE) is a node of the multi-dimensional hypercube. All PEs and all links in the hypercube are homogeneous, i.e., PEs have the same processing speed and links have the same communication

0-7695-1010-8/01 $10.00 © 2001 IEEE

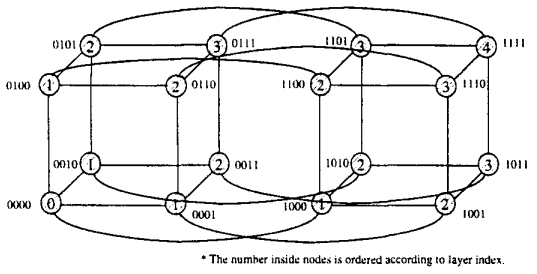


Figure 1. A four-dimensional hypercube

speed. We assume that all PEs are equipped with front-end, and hence, they can carry out computation and communication simultaneously. For a d-dimensional hypercube, there are 2^d processors. We label the nodal address of each PE using a binary string from the range $[0, 2^d - 1]$ [5]. The root node p_0 is the originator of the load.

We are concerned on the problem of how to optimally distribute the load among these $n = 2^d$ processors in a d-dimensional hypercube such that the total processing time of the divisible task is minimum. To be specific, we describe the load scheduling process more formally as follows. At time 0, a divisible job resides at the root node p_0. A load fraction α_0 is retained and processed locally. The rest of the load is divided into d parts and transmitted to its neighbors. The immediate neighbors of the root node intercept some load fractions to be processed locally and re-transmit the rest of the load to the idle processors at a one-hop distance. This receiving, intercepting and retransmitting process is continued until the last idle processor numbered $(2^d - 1)$ is assigned a loadd fraction to be processed. Note that, during this load propagation process, we assume that the data communication can adopt scattering and gathering paradigms. As shown in Figure 1, we refer to the set of processors with the same Hamming distance of the PE label from the originator as a layer. We denote these layers as layer $0, 1, ..., i, ..., d$ with the root node at the layer 0 and the last processor numbered $2^d - 1$ at the layer d. For convenience, we denote layer i as Y_i. Since it is a homogeneous system, we denote each processor at layer Y_i as p_i. Thus, the problem is then to determine the optimal size of load fractions that are assigned to each processor to achieve the minimum processing time of the total load. We use the following notations and definitions throughout this paper.

Let L be the total amount of load originating at p_0 for processing and L_i be the load fraction transmitted to each processor at layer Y_i for processing. We denote α_i as the percentage of the load fraction L_i intercepted at each processor at layer Y_i. As in the literature [2], the time taken to compute a unit load by each processor and time taken to transmit a unit load on each link are denoted as E and C, respectively.

Thus, using the above notations, we see that the communication time to transmit a fraction of load $\alpha_i L_i$ to a one-hop neighbor is given by $\alpha_i L_i C$, and the computation time of this load fraction by a processor at layer Y_i is given by, $\alpha_i L_i E$.

3. Optimal Solutions with infinite-size buffers

In this section, we shall derive the optimal solution when the buffer size at each processor is infinite and present analyses for the cases with and without granularity constraint separately.

3.1. Optimal solutions without granularity constraint

In this section, we shall first present the optimal load scheduling strategy without finite-size buffers and granularity constraint and derive the solutions for the optimal processing time. Readers are referred to the details of the similar derivation in [3]. The load propagation and computation process from layer Y_0 to layer Y_d is illustrated by means of a timing diagram as shown in Figure 2. Note that, in

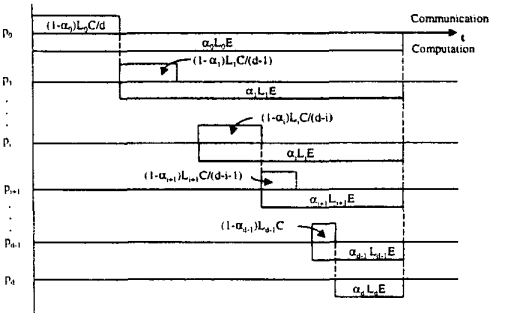


Figure 2. Timing diagram for a d-dimensional hypercube: without buffer constraint

Figure 2, we only show the time behavior of one processor p_i in layer Y_i since all processors and links are identical. We follow the optimality principle in DLT literature [1, 2], i.e., all the processors involved in processing shall stop processing at the same time to achieve the minimum processing time of the total load. As proven in [3], in each layer Y_i of d-dimensional hypercube there are $\binom{d}{i}$ processors, $i = 0, ..., d$. Each processor at layer Y_i can be transferred a load fraction from i processors at layer Y_{i-1} and it can transmit rest of load fraction to $(d - i)$ still idle processors at layer Y_{i+1}. Hence, we have the following relation-

ship between the load distributed to successive layers Y_i and Y_{i+1}.

$$L_{i+1} = \frac{(1-\alpha_i)L_i}{d-i}(i+1),\ i = 0, 1, ..., d-1. \quad (1)$$

In Figure 2, we observe that the processor at layer Y_i intercepts a load fraction $\alpha_i L_i$ to be processed locally and transmit the rest load $(1-\alpha_i)L_i$ divided equally to the still idle $(d-i)$ processors at layer Y_{i+1}. Thus, the computation time of this processor is $\alpha_i L_i E$ and the communication time is $(1-\alpha_i)L_i C/(d-i)$.

From the timing diagram shown in Figure 2, we obtain the following recursive equations.

$$\alpha_i L_i E = \alpha_{i+1} L_{i+1} E + \frac{(1-\alpha_i)L_i C}{d-i},\ i = 0, 1, ..., d-1. \quad (2)$$

We know that the final node at layer Y_d will accept all the newly arriving load fractions, thus $\alpha_d = 1$. Replacing (2) with (1), we obtain the following recursive solutions for α_i.

$$\alpha_i = \frac{1}{1 + \frac{(d-i)E}{(i+1)\alpha_{i+1}E + C}},\ i = 0, 1, ..., d. \quad (3)$$

From the above recursive equations, we can obtain all the α_i, $i = 0, 1, ..., d$. Since L_0 is equal to the total load L, using the recursive equations (1), we obtain all L_i, $i = 0, 1, .., d$. Thus, the optimal solution of load distribution is described as follows: each processor on layer Y_i receives load fraction L_i, intercepts $\alpha_i L_i$, and transmits the rest, $(1-\alpha_i)L_i$, to idle processors on layer Y_{i+1}. From Figure 2, we see that the final processing time is given by $\alpha_0 L_0 E$.

Lemma 1 *According to the load distribution strategy proposed, load fraction assigned to each PE at layer i is always greater than load fraction assigned to each PE at layer $i+1$, i.e., $\alpha_i L_i > \alpha_{i+1} L_{i+1}, i = 0, ..., d-1$.*

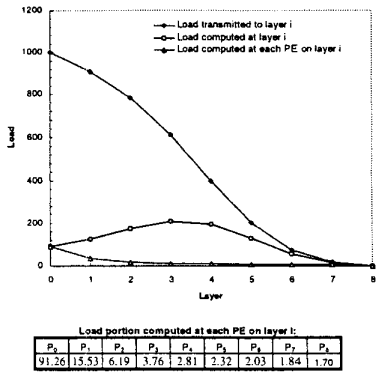
Proof: Rewriting (2), we obtain,

$$\alpha_i L_i = \alpha_{i+1} L_{i+1} + \frac{(1-\alpha_i)L_i C}{(d-i)E} \quad (4)$$

Since $1-\alpha_i > 0$ and $d-i > 0$, we immediately find that $\alpha_i L_i > \alpha_{i+1} L_{i+1}, i = 0, 1, ..., d-1$. ■

Lemma 1 shows the monotonicity property of load distribution so that each processor on layer Y_i will process more load than each processor on its successive layers. We will use this monotonicity property in our later analysis.

Example 1 *Consider an 8-dimensional hypercube with the following speed parameters: $E = 3.0$ s/Mb and $C = 2.0$ s/Mb, and $L = 1000Mb$.*



Load portion computed at each PE on layer i:

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
91.26	15.53	6.19	3.76	2.81	2.32	2.03	1.84	1.70

Figure 3. Load on layers and PEs with respect to layer index for Example 1

Using recursive equations (2) and (3), the optimal schedule is $\alpha_0 = 0.09126$, $\alpha_1 = 0.13671$, $\alpha_2 = 0.22091$, $\alpha_3 = 0.34489$, $\alpha_4 = 0.49141$, $\alpha_5 = 0.63963$, $\alpha_6 = 0.77635$, $\alpha_7 = 0.89655$, $\alpha_8 = 1.0$, and $L_0 = 1000.0Mb$, $L_1 = 908.7Mb$, $L_2 = 784.5Mb$, $L_3 = 611.2Mb$, $L_4 = 400.4Mb$, $L_5 = 203.6Mb$, $L_6 = 73.4Mb$, $L_7 = 16.4Mb$, $L_8 = 1.7Mb$. Then, $T^*(\alpha, d) = \alpha_0 L_0 = 273.8$ *sec.*

From previous analysis, we know that load assigned to layer i is L_i, load computed at each PE of layer i is $\alpha_i L_i$, and load computed at layer i is $\alpha_i L_i \binom{d}{i}$. These load distribution versus layer index are illustrated in Figure 3, which testifies Lemma 1. However, the load size computed at each layer increases from layer 1 to layer 3 and decreases from layer 3 to layer 8. This behavior is due to the fact that the number of PEs at layer i is $\binom{d}{i}$.

3.2. Scheduling strategies with granularity constraint

Most of literature in DLT assume that the input load is infinitely divisible. In a realistic situation, tasks could be fine grained or coarse grained, however they cannot be infinitely divided. Also, due to computation and communication overheads, fine-grained partitions of tasks may deteriorate system performance. Thus, most of the real life applications impose a constraint on the extent to which a given task may be partitioned. In this paper, we shall set a threshold, say, δ, as a minimum size that can be assigned to any processor. In this section, we will address the scheduling problem with limited granularity constraint, i.e. all load fractions shall be greater than δ. Now, we propose a simple algorithm as follows.

After iterative check, we obtain the largest reachable

Algorithm A:
Let $k = 0$; bSucess=FALSE;
Using equations (1) and (3), we obtain all α_i and $L_i, i = 0, 1, ..., d$.
FOR ($i = d$; $i <= 1$; $i = i - 1$) {
  IF ($\alpha_i L_i \geq \delta$) {
    $k = i$;
    bSuccess=TRUE;
    BREAK;
  }
  k=i; set $\alpha_k = 1$
  Using (1) and (3), Update(α_i, L_i) for $i = 0, 1, ..., k$;
}
IF (bSuccess)
  We have obtained the sub-optimal load schedule α_i and L_i, for $i = 0, 1, ..., k$. The maximum reachable layer is k.
ELSE
  Fails to find a sub-optimal solution.

layer k when all $\alpha_i L_i < \delta$, for $i = k + 1, ..., d$. Thus, since layer Y_k is the last layer of load propagation, we set $\alpha_k = 1$. By applying equations (1) and (3) among the range $i = 0, 1, ..., k$, we obtain the sub-optimal load schedule, α_i and L_i for $i = 0, 1, ..., k$.

Example 2 *Consider an 8-dimensional hypercube system with the following parameters: $E = 2$ s/Mb, $C = 3$ s/Mb, and $L = 1000Mb$. Let $\delta = 2$ Mb and $\delta = 4$ Mb respectively. Figure 4 shows the performance of Algorithm A.*

As shown in Figure 4, we see that the basic tendency of the load distribution on each layer or each PE is very similar to that in Example 1. However, due to the granularity constraint, the maximum reachable layer is $k = 6$ for $\delta = 2Mb$ and $k = 4$ for $\delta = 4Mb$. We observe that in the case $\delta = 2Mb$, each PE on the layer Y_6 (last layer) is assigned $2.16Mb$ to compute, which satisfies our granularity constraint $min\{\alpha_i L_i\} \geq \delta$. When $\delta = 4Mb$, each PE on the layer Y_4 (last layer) is assigned $4.24Mb$ to compute, which satisfies our granularity constraint. Naturally, the maximum reachable layer for the case with coarse-grained tasks (Figure 4 (b)) is smaller than the maximum reachable layer for fine-grained tasks (Figure 4 (a)).

4. Optimal Solutions with Finite-size Buffers

For this case, we cannot directly use (3) and (1) to obtain an optimal load distribution. Due to the finite-size buffer constraint, some processors cannot accommodate the optimal load fraction obtained for the case without buffer size constraints. However, we will consider these solutions in our new load scheduling strategy. We shall first introduce some new notations.

Let $B_i^{(k)}$ be the buffer capacity available at each processor of layer Y_i at the k-th iteration, with $B_i^{(0)} = B$ as the initial buffer capacity, $i = 0, 1, ..., d$. Let the load fraction transmitted to each processor at layer Y_i for processing at the k-th iteration, $i = 0, 1, ..., d$, be denoted as $L_i^{(k)}$. Let ℓ_i^* be the final load fraction transmitted to each processor at layer Y_i for processing, $i = 0, 1, ..., d$ and let α_i be the percentage of the load fraction L_i intercepted at each processor at layer Y_i at the k-th iteration, $i = 0, 1, ..., d$. Further, we denote the load portion left undistributed at the k-th iteration, initially $R^{(0)} = L$ as $R^{(k)}$ and $X^{(k)}$ be the load portion distributed at the k-th iteration, initially $X^{(0)} = 0$.

In Table 1, we present the proposed algorithm in pseudo-codes. We refer to this algorithm as *Incremental Balancing Strategy* (IBS) as the strategy attempts to balance the load using the optimal solutions in an incremental fashion to reach the optimal solution.

In essence, at every iteration, the algorithm tries to take advantage of the property that the optimal processing time is achieved when all the processors involved in processing stop computing at the same time instant for the case with infinite size buffers. Note that, at k-th iteration, we consider all the the processors with vacant buffers. The expression for the amount of load that is processed at the k-th iteration, namely, $X^{(k)} = \binom{d}{q} L_q^{(k)}$, where $L_q^{(k)} = min(B_i^{(k)}/\alpha_i^{(k)})$, guarantees that all the respective processors with vacant buffers can accommodate the optimal load portion assigned to them. This means that these processors will stop processing at the same time instant. Besides, note that those buffers, whose size satisfies the RHS of the expression $L_q^{(k)}$ mentioned above, will be filled at this iteration. One of the advantages of this algorithm is in finite step convergence, and this is essentially due to the choice of $L_q^{(k)}$. Later, we will show that the algorithm takes at most $(d + 1)$ iterations to schedule/process the entire load.

Example 3 *Consider an 8-dimensional hypercube with the following speed parameters: $E = 3.0s/Mb$ and $C = 2.0s/Mb$. The buffer size at each node of the hypercube is 10.0 Mb and $L = 1000$ Mb. Refer to Table 2 for details.*

From the results shown in Table 2, we see that the optimal processing time is given by $T^*(\alpha, d) = 301.5$. The optimal load fractions are obtained after three iterations of IBS algorithm, $\ell_0^* = 1000.0Mb$, $\ell_1^* = 123.75Mb$, $\ell_2^* = 32.5Mb$, etc. Thus, the root node p_0 receives $1000Mb$ load, intercepts $\alpha_0 * \ell_0^* = 10.0$ to be computed locally and transmits load fraction $\ell_1^* = 123.75Mb$ to each processor at next layer Y_1. Each processor P_1 at layer Y_1 intercepts $\alpha_1^* \ell_1^* = 10.0$ to be computed locally and transmits the rest

Table 1. Incremental balancing algorithm IBS

Initialization:

$B_i^{(0)} = B$, $L_i^{(0)} = L$, $\alpha_i^{(0)} = 0$, $R^{(0)} = L$, $X^{(0)} = 0$, q=0.
/* q is the index of the maximum filled layer */
/* schedulability check */
if($L > B \sum_{i=0}^{d} \binom{d}{i}$) exit with error message "no enough buffers";
if($L = B \sum_{i=0}^{d} \binom{d}{i}$) exit with message "optimal schedule is to fill all the buffers exactly";
set k=1; bStop=FALSE; /* k is the index of iteration */

Do {
Phase I: *Resetting the variables*
$X^{(k)} \leftarrow X^{(k-1)}$, $B_i^{(k)} \leftarrow B_i^{(k-1)}$.
Phase II: *Scheduling*
Through (3) and $\alpha_d = 1$, we obtain the $\alpha_i^{(k)}$. Taking the buffer constraints into consideration, we determine $L_q^{(k)} = min(B_i^{(k)}/\alpha_i^{(k)}) = B_q^{(k)}/\alpha_q^{(k)}$ for all $B_i^{(k)} > 0$. Thus, we obtain the load portion distributed at this iteration, $X^{(k)} = \binom{d}{q} L_q^{(k)}$.
Phase III: *Update*
If ($X^{(k)} \geq R^{(k)}$)
{ let $X^{(k)} \leftarrow R^{(k)}$; $L_q^{(k)} = X^{(k)}/\binom{d}{q}$; bStop=TRUE;}
else
{ Load left unprocessed is $R^{(k)} \leftarrow R^{(k)} - X^{(k)}$; }
for (for all $q+1 \leq i \leq d$)
{ $L_i^{(k)} = M(q, i-1) L_q^{(k)}$; }
for (for all $0 \leq i \leq q-1$)
{ $L_i^{(k)} = X^{(k)}/\binom{d}{i}$; $\alpha_i^{(k)} = 0$; }
for (for all $q \leq i \leq d$)
{ $B_i^{(k)} = B_i^{(k)} - \alpha_i^{(k)} L_i^{(k)}$;}
if (bStop) break; /*goto Optimal Solution*/
q=q+1;

Set $k = k + 1$
}**while**($k \leq d + 1$)

Optimal Solution:
Total number of iterations is $r = k$;
The optimal schedule is given by { ℓ_i^*} and { α_i^*}.
$\ell_i^* = \sum_{k=1}^{r} L_i^{(k)}$, $\alpha_i^* = \sum_{k=1}^{r} \alpha_i^{(k)} L_i^{(k)}/\ell_i^*$, $i = 0, 1, ..., d$.
$T^* = \sum_{j=0}^{d-1} (1 - \alpha_i^*) \ell_i^* C/(d-i) + \alpha_d^* \ell_d^* E$

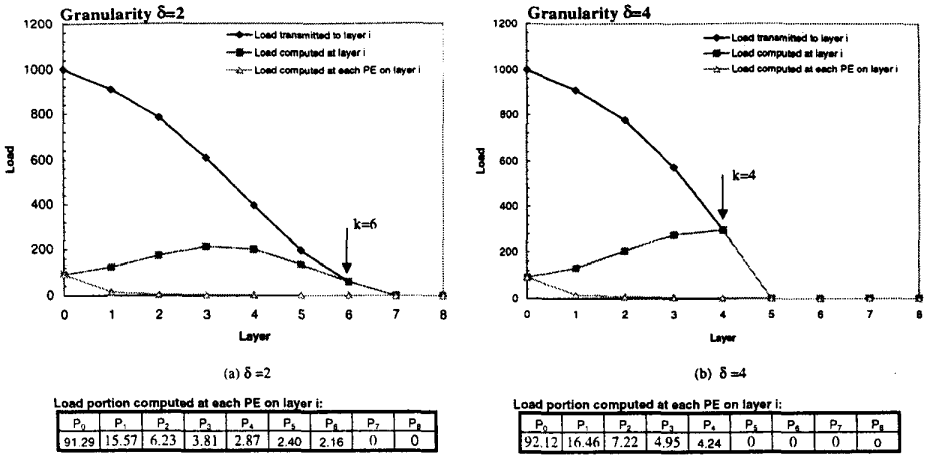


Figure 4. Load on layers and PEs with respect to layer index for Example 2

Table 2. The results of each iteration for Example 3

k	$B_0^{(k)}$	$B_1^{(k)}$	$B_2^{(k)}$	$B_3^{(k)}$	$B_4^{(k)}$	$B_5^{(k)}$	$B_6^{(k)}$	$B_7^{(k)}$	$B_8^{(k)}$	$X^{(k)}$
0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	0
1	0.0	8.3	9.3	9.6	9.7	9.7	9.8	9.8	9.8	109.6
2	0.0	0.0	6.0	7.6	8.4	8.5	8.74	8.8	8.9	485.6
3	0.0	0.0	2.8	5.6	6.7	7.3	7.6	7.9	8.0	404.8
ℓ_i^*	1000.0	123.75	32.5	12.66	6.64	4.22	3.04	2.38	1.97	
α_i^*	0.010	0.0808	0.221	0.345	0.491	0.640	0.776	0.896	1.0	
T_i	30.0	80.0	301.5	301.5	301.5	301.5	301.5	301.5	301.5	

$(1 - \alpha_1^*)\ell_1^*$ to processors at the next layer as described in Section 3.

Load distribution results of Example 3 are illustrated in Figure 5. Compared to Figure 3, we see, Figure 5 shows very similar curves of load distribution. This load distribution pattern indicates that our proposed algorithm follows the optimality principle. However, in Figure 5, due to the finite-size buffer constraint, each PE has only limited buffer to accommodate assigned load. Thus, load computed at p_0 is only $10.0Mb$ other than $91.26Mb$ in Figure 3.

A number of interesting points can be observed at this juncture. Firstly, we find that at every iteration (with the exception of the last iteration), there is at least one buffer that is completely filled and hence, that processor is not available in the successive iterations. Also, it is to be noted that in general, even in the last iteration, the respective buffers may be filled. This is not a coincidence, but an intrinsic property of the IBS algorithm, which will be shown in the next section. The algorithm utilizes the available buffer at a site to the fullest and best possible extent and in the last iteration, only the processors p_3 to p_5 alone participate in processing the load, as other buffers with the rest of the processors are completely filled. Further, another interesting observation is that the processors p_3 to p_5 stop processing at the same time yielding the optimal processing time. In Table 2, the last column specifies the total amount of load consumed at every iteration by all the participating processors. Each row, on the other hand, specifies the amount of buffer that is left unfilled (available for the next iteration) at each site.

4.1. Analysis of the IBS algorithm

In this section, we shall present detailed analysis and proofs of several significant properties of the IBS algorithm.

Lemma 2 *At the end of k-th iteration, if there is any excess load that is left un-processed, i.e., if $R^{(k)} > 0$, then there is a layer whose buffers are just filled at this iteration.*

Proof: From the IBS algorithm, we let $L_q^{(k)} = min\{\frac{B_i^{(k)}}{\alpha_i^{(k)}}\}$. There is at least one equality, say $L_q^{(k)} = \{\frac{B_q^{(k)}}{\alpha_q^{(k)}}\}$. The

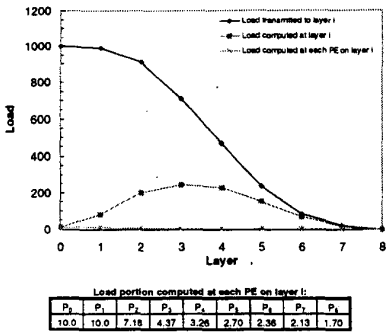


Figure 5. Load distribution on layers and PEs vs. layer index for Example 3

buffers at layer Y_q will be filled exactly. Hence the proof. ■

This lemma signifies the fact that in every iteration, until all the load is processed there is at least one buffer that will be filled completely. This means that at any iteration, if needed, the algorithm uses the buffers to the fullest possible extent as demonstrated in Example 3.

Lemma 3 *The total number of iterations before the algorithm terminates, denoted as r, is no more than (d + 1), $r \leq d+1$.*

Proof: From Lemma 2, we know that there is at least one layer whose buffers filled at each iteration. In the d-dimensional hypercube system as defined, there are totally $(d+1)$ layers. Under the condition that the total load is less than the sum of buffers in the system, $L < B\sum_{i=0}^{d}\binom{d}{i}$, the load will never completely fill all the buffers. Since there is one layer whose buffers are filled at each iteration, the total iterations required to obtain the final optimal solution is no more than $(d+1)$. Hence the proof. ■

Lemma 3 guarantees the feasibility aspect of the algorithm and derives the complexity of the proposed algorithm.

Lemma 4 *At the end of k-th iteration, all the processors of layer Y_i ($i = k, k+1, ..., d$) stop processing at the same time.*

Proof: The proof is by using induction on the iterations. We omit the details due to space constraints.

Lemma 5 *The IBS algorithm guarantees that there is no load left unprocessed when it terminates. Besides, the final processing time is determined by the processors with unused buffers and these processors stop processing at the same time.*

Proof: From Lemma 3, the algorithm terminates in at the most $(d+1)$ iterations. Applying Lemma 4 to the final iteration, we see that the proof follows immediately. Hence the proof. ■

Conjecture 1 *The IBS algorithm produces the optimal schedule.*

The proof of the above claim is quite intuitive and evident from the construction of the algorithm and also from the properties shown through lemmas 1 through 5. However, in general, a formal proof is yet to be attempted.

4.2. Scheduling strategies with granularity constraint

For this case, we propose a revised algorithm, Algorithm A′.

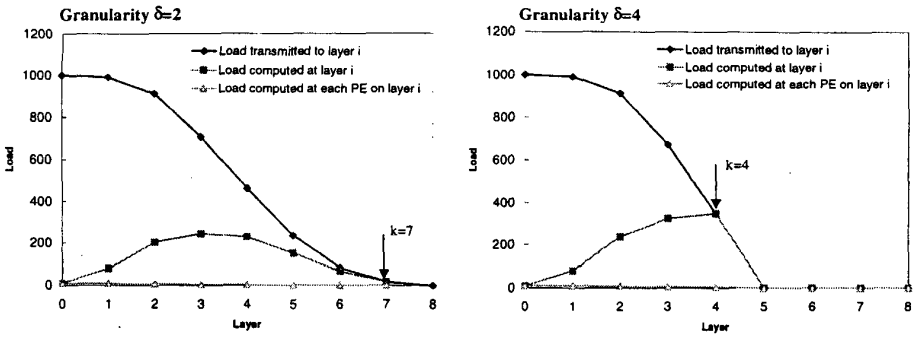
Algorithm A′:

Let $k = 0$; bSucess=FALSE;
Following IBS algorithm, we obtain all α_i and L_i, $i = 0, 1, ..., d$.
FOR ($i = d$; $i <= 1$; $i = i - 1$) {
  IF ($\alpha_i L_i \geq \delta$) {
    $k = i$;
    bSuccess=TRUE;
    BREAK;
  }
  k=i; set $\alpha_k = 1$;
  Following IBS algorithm, Update(α_i, L_i) for $i = 0, 1, ..., k$;
}
IF (bSuccess)
  We have obtained the sub-optimal load schedule α_i and L_i, for $i = 0, 1, ..., k$. The maximum reachable layer is k.
ELSE
  Fail to find a sub-optimal solution.

In the following, we show an example applying Algorithm A′. The results are shown in Figure 6.

Example 4 *Consider an 8-dimensional hypercube system with the following parameters: $E = 2$ s/Mb, $C = 3$ s/Mb. The buffer size at each node of the hypercube is 10.0 Mb and $L = 1000Mb$. Let $\delta = 2$ Mb and $\delta = 4$ Mb respectively. Following the Algorithm A′, we obtain the sub-optimal solutions for each case as illustrated in Figure 6.*

As shown in Figure 6, we see that the the behavior of the load distribution on each layer or each PE is very similar



Load portion computed at each PE on layer i:

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
10	10	7.18	4.37	3.27	2.71	2.37	2.17	0

Load portion computed at each PE on layer i:

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
10	10	8.47	5.80	4.97	0	0	0	0

Figure 6. Load on layers and PEs with respect to layer index for Example 4

to that in Example 3. However, due to the granularity constraint, the maximum reachable layer is $k = 7$ for $\delta = 2Mb$ and $k = 4$ for $\delta = 4Mb$. Compared to Example 2 and Figure 4, we observe that the maximum reachable layer ($k = 7$) for the case with finite-size buffer constraint is greater than that ($k = 6$) in the case without buffer constraint. This phenomenon is due to the influence of both the buffer and granularity constraints. Because of buffer constraint, the load computed at processors on layer 0 and layer 1 is less, thus pushing more load to the other layers. From Figure 6, we notice that the load fractions on each PE satisfy both constraints, which convincingly demonstrates the feasibility of our algorithms.

5. Discussions and Conclusions

The problem of scheduling a divisible load onto a hypercube cluster of processors with finite-size buffers and finite load granularity was investigated in this paper.

To find the optimal schedule under the constraint of finite-size buffers, we proposed an algorithm referred to as IBS algorithm. Several interesting properties of IBS algorithm were presented and proven in lemmas. In addition, under the constraint of both finite-size buffer and finite granularity, we devise an algorithm to determine a sub-optimal solution that satisfies both the constraints. We observe that, in *Algorithm A* and *Algorithm A'*, we use an iterative check to find the sub-optimal solution. An open problem is to rigorously prove the optimality of IBS algorithm and also to see the time performance on other topologies.

References

[1] Barlas, Gerassimos D., " Collection-Aware Optimum Sequencing of Operations and Closed-Form Solutions for the Distribution of a Divisible Load on Arbitrary Processor Trees", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No. 5, May, 1998, pp. 429-441.

[2] Bharadwaj, V., D. Ghose, V. Mani, and T.G. Robertazzi, " Scheduling Divisible Loads in Parallel and Distributed Systems", *IEEE COmputer Society Press*, Los Almitos, California, 1996.

[3] Blazewicz, J., and M. Drozdowski, "Scheduling divisible jobs on hypercubes", *Parallel Computing*, Elsevier Science, 1995.

[4] Cheng, Y. C., and T. G. Robertazzi, "Distributed Computation with Communication Delays", *IEEE Transactions on Aerospace and Electronic Systems*, **24**,700-712, 1988.

[5] Hwang, K., "Advance Computer Architecture: Parallelism, Scalability, Programmability", *McGraw-Hill*, 1993.

[6] Li, X., V. Bharadwaj, and C.C. Ko, "Divisible Load Scheduling on Single-level Tree Networks with Finite-size Buffers", (to appear) *IEEE Transactions on Aerospace and Electronic Systems*, Vol.37, Jan. 2001.