Syracuse University
SURFACE

L.C. Smith College of Engineering and Computer Science - Former Departments, Centers, Institutes and Projects

L.C. Smith College of Engineering and Computer Science

1-1-1993

A Probabilistic Analysis of a Locality Maintaining Load Balancing Algorithm

Kishan Mehrotra
Syracuse University, mehrotra@syr.edu

Sanjay Ranka
Syracuse University

Jhy-Chun Wang
Syracuse University

Follow this and additional works at: http://surface.syr.edu/lcsmith_other

Part of the Computer Sciences Commons

Recommended Citation
Mehrotra, Kishan; Ranka, Sanjay; and Wang, Jhy-Chun, "A Probabilistic Analysis of a Locality Maintaining Load Balancing Algorithm" (1993). *L.C. Smith College of Engineering and Computer Science - Former Departments, Centers, Institutes and Projects.* Paper 26.
http://surface.syr.edu/lcsmith_other/26

This Article is brought to you for free and open access by the L.C. Smith College of Engineering and Computer Science at SURFACE. It has been accepted for inclusion in L.C. Smith College of Engineering and Computer Science - Former Departments, Centers, Institutes and Projects by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.

A Probabilistic Analysis of a Locality Maintaining Load Balancing Algorithm

Kishan Mehrotra, Sanjay Ranka[1], and Jhy-Chun Wang[1]
4-116 Center for Science and Technology
School of Computer and Information Science
Syracuse University
Syracuse, NY 13210

[1]This work was supported in part by NSF under CCR-9110812 and DARPA under contract No. DABT63-91-C-0028. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

Abstract

This paper presents a simple load balancing algorithm and its probabilistic analysis. Unlike most of the previous load balancing algorithms, this algorithm maintains locality. We show that the cost of this load balancing algorithm is small for practical situations and discuss some interesting applications for data remapping.

Index Terms - Data locality, irregularity, load balancing, mapping, probabilistic analysis.

1 Introduction

In parallel computing, it is important to map the program such that the total execution time is minimized. Experience with parallel computing has shown that a 'good' mapping is a critical part of executing a program on such computers. This mapping can be typically performed statically or dynamically.

For most regular and synchronous problems [9], this mapping can be performed at the time of compilation by giving directions in the language to decompose the data and its corresponding computations (based on the owner computes rule). We are currently developing a compiler for Fortran D, which provides a rich set of such directives [5]. Load balancing and reduction of communication are two important issues for achieving load balancing. The directives of Fortran D can be used to provide such a mapping for a large class of regular and synchronous problems.

For some other class of problems, which are irregular in nature, achieving good mapping is considerably more difficult. Further, the nature of this irregularity may not be known, and can be derived only at run time. Many problems can be characterized as a discrete model of a physical system, and a set of values are to be calculated at every domain point of the system [15]. The mapping of such problems entails mapping of regions of model domain to each processor. The computational work associated with each subdomain may change over a period of time and hence the load on each processor may become unbalanced. For many problems, the computations may be characterized as a series of phases. The output of each phase acts as an input for the next phase. Although the input may have uniform pattern, the output may be nonuniform. For example, computer vision requires the conversion of image (low level structure) into higher level structures. The processing passes through several phases. The following are some of the low-level tasks, where the output of phase 1 would be used as a input to phase 2 or phase 3 or both:

1. The image is converted into a set of edges by application of Sobel operator [1](to give an edge image).

2. The edge image can be used to detect lines or circles in the image.

3. Multiple images can be used to perform stereo effect for detection of motion or distance of the object.

A typical parallelization of these tasks would require partitioning of the input image. Assume that we have a image of size $N \times N$ distributed on p processors such that each processor gets a $\frac{N}{p} \times N$ rectangular block. (We note that it may be useful in some cases to divide the image in each processor such that each node gets a $\frac{N}{\sqrt{p}} \times \frac{N}{\sqrt{p}}$ square block. However, we restrict ourselves to the previous mapping). The number of edges in each partition in general will not be equal. However, phase 2 or 3 may require locality of edges. In such cases the load needs to be balanced in a fashion that each node has equal number of edges (assume that the computation depends on the number of edges).

In such cases a remapping needs to be performed in order to achieve load balancing and have potential improvement in performance. There are many algorithms described in the literature for mapping irregular problems (e.g. [4, 10, 12]). These algorithms perform the mapping statically and are very time consuming. For many problems this is acceptable, as the structure of the problem does not change over its execution. However, they are prohibitive for a large class of applications. There are several algorithms proposed in the literature for balancing the load at run time [6, 11, 14, 18]. However, these algorithms shuffle data around in a fashion that locality between data items is no longer maintained. For applications possessing some natural locality, *i.e.*, the computations utilize data items which have some sense of proximity, shuffling of the data to balance the load will, in general, lead to a greater and irregular communication and may significantly reduce the advantages of having the load balance.

In this paper, we analyze a simple load balancing algorithm for irregular problems. A similar algorithm has been described in [13] for load balancing for fine grained hypercube machines. We show that if irregularity is such that the computation points are distributed with a certain class of distributions and the granularity (number of points per processor) is reasonably high, then the cost of this load balancing is nominal and reduces to a simple shift algorithm. Further the load balancing algorithm

maintains locality which is one of the desirable features. We give some simple applications of the load balancing algorithm which could be used in several domains.

The rest of this paper is organized as follows. Section 2 describes several different versions of the load balancing algorithm. Section 3 presents average analyses of the load balancing algorithms. These algorithms are developed in an architecture independent fashion using collective communication primitives with reasonable assumptions about the cost of these primitives. This makes them suitable for a wide variety of architectures. Section 4 presents a simple application. Finally, conclusions are presented in Section 5.

2 Load Balancing Algorithm

Let the data which is useful in kth processor, P_k, be given by array $aLoc_k(0..X_k-1)$, where X_k represents the number of useful elements in P_k, where $k = 0, \cdots, n-1$. We assume that the data in each local array is sorted in order of locality.

The load balancing algorithm is given in Figure 1. The following variables are used in the algorithm:

- prefix sum $Y_k = \sum_{i=0}^{k-1} X_i$ for $k = 1, \cdots, n-1$, and $Y_0 = 0$.
- average number of useful elements $\overline{X} = \frac{1}{n}\sum_{i=0}^{n-1} X_i$. We assume that $\overline{X}$ is an integer (we make this assumption for ease of presentation). The algorithm can be easily modified when this is not satisfied.
- $G_k(i)$ represents $aLoc_k(i)'s$ corresponding global index, $G_k(i) = Y_k + i,\ 0 \leq i \leq X_k - 1$.
- $packet_i^k$ contains data elements which should be moved from processor P_k to P_i. Let $lb_i^k = \max\{i\overline{X}, Y_k\}$ and $ub_i^k = \min\{(i+1)\overline{X} - 1, Y_k + X_k - 1\}$, then if $lb_i^k > ub_i^k$, $packet_i^k = \phi$, otherwise $packet_i^k = \{aLoc_k(j) \mid G_k^{-1}(lb_i^k) \leq j \leq G_k^{-1}(ub_i^k)\}$, $where\ G_k^{-1}(i) = i - Y_k$.

Load Balancing Algorithm:
For processor P_k, $0 \leq k \leq n-1$, *parallel do*

1. $Y_k = Parallel_Sum_Prefix(X_k)$;
2. $\overline{X} = \frac{1}{n} \cdot Parallel_Sum(X_k)$;
3. $Rshift_k = \lfloor \frac{G_k(X_k - 1)}{\overline{X}} \rfloor - k$;
   $Lshift_k = k - \lfloor \frac{G_k(0)}{\overline{X}} \rfloor$;
4. $Max_L_Shift = Parallel_Max(Lshift_k)$;
   $Max_R_Shift = Parallel_Max(Rshift_k)$;
5. *call Data_Movement()*;

Figure 1: Load Balancing Algorithm

- $Lshift_k$ ($Rshift_k$) represents the maximum distance of left (right) shift P_k will perform. It should be noted that $Lshift_k$ and $Rshift_k$ could be negative (implying that this shift takes place on the opposite direction, it also represents the minimum shift in that direction). Further $Lshift_0 = 0$ and $Rshift_{n-1} = 0$.
- Max_L_Shift (Max_R_Shift) represents the maximum distance of left (right) shift among all processors.

In this paper, we analyze our algorithms in architecture independent fashion. We assume a store-and-forward message passing approach for calculating the complexity of the communication. However, our algorithms are developed using collective communication, which could utilize wormhole or cut-through routing [7]. Further, the main results of our paper are not dependent on the above choice. We assume that a linear array can be efficiently embedded in the architecture. This is true for popular architectures like meshes, toruses, and hypercubes [16]. The time to send a message

of size S from any node to a neighbor node is assumed to be $O(\tau + \varphi S)$, where τ represents the set up cost and φ represents the inverse of the data transfer rate. For efficiency reasons our algorithms require efficient evaluation of parallel prefixes. Prefix operations are provided in hardware on CM-5 [20], it is expected that it would be available on most future computer architectures.

In this paper we propose several schemes for data movement, each approach may be suitable for a particular system architecture. The time required for step 1, 2, and 4 (Figure 1) is upper bounded by the time required for parallel prefix. Step 3 can be completed in $O(1)$. We develop several algorithms for step 5. All three algorithms assume that a linear array can be embedded in the given architecture.

2.1 Approach 1

In this approach (Figure 2), each processor P_k first concatenates all packets it needs to send to its left hand side processors (*i.e.* $P_i,\ i < k$). At each stage, P_k shifts its packets to P_{k-1} and receives packets from P_{k+1}, P_k then accepts and removes the packets which are targeted to it from the packets it received. The stage will be repeated until all packets reach their final destination. The right shift operation will follow the same procedure, but in other direction.

Assume S represents the maximum size of packets (in terms of data elements) which would be left shifted among processors, also let D represent the longest left shift distance among processors. Then in the worst case one processor may contain as many as DS data needed to be left shifted, so the time takes to complete the left shift process would be

$$(\tau + D\varphi S) + (\tau + (D-1)\varphi S) + \cdots + (\tau + \varphi S)$$
$$= D\tau + \frac{D(D+1)}{2}\varphi S.$$

So the worst case time complexity of this approach is $O(D\tau + D^2\varphi S))$. This approach is geared towards architectures which utilize store and forward communication method.

procedure Data_Movement();

1. Let $L_packets_k = \cup_{i=k-Lshift_k}^{k-1} packet_i^k$;
   /* concatenate left-shift data in one packet */

2. $for \;\; i = 1 \;\; to \;\; Max_L_Shift \;\; do$

   (a) P_k send $L_packets_k$ to P_{k-1};

   (b) P_k receive $L_packets_{k+1}$ from P_{k+1};

   (c) Let $L_packets_k = L_packets_{k+1} - packet_k^j, \;\; k+1 \leq j \leq n$;

3. Let $R_packets_k = \cup_{i=k+1}^{k+Rshift_k} packet_i^k$;
   /* concatenate right-shift data in one packet */

4. $for \;\; i = 1 \;\; to \;\; Max_R_Shift \;\; do$

   (a) P_k send $R_packets_k$ to P_{k+1};

   (b) P_k receive $R_packets_{k-1}$ from P_{k-1};

   (c) Let $R_packets_k = R_packets_{k-1} - packet_k^j, \;\; 1 \leq j \leq k-1$;

Figure 2: Data Movement: Approach 1

The other way to perform the complexity analysis is to assume that the maximum amount of data to be sent by any processor is X. In that case the complexity is $O(D(\tau + X\varphi))$.

2.2 Approach 2

In this approach (Figure 3), each processor P_k initializes a vector $send_k$, where $send_k[i] = 1$ if P_k needs to send packets to P_i, otherwise $send_k[i] = 0$. All processors then participate in $Parallel_Sum(send[\,])$, which will return a vector $receive[\,]$ with

procedure Data_Movement();

1. Let $send_k[1..n] = 0$;

2. $for\ \ i = 1\ \ to\ \ n\ \ do$
   $\quad if\ \ packet_i^k \neq \phi\ \ then\ \ send_k[i] = 1$;

3. $receive_k[1..n] = Parallel_Sum(send_k[1..n])$;

4. $for\ \ i = 1\ \ to\ \ n\ \ do$
   $\quad if\ \ packet_i^k \neq \phi\ \ then\ \ send\ \ packet_i^k\ \ to\ \ P_i$;

5. $for\ \ i = 1\ \ to\ \ receive_k[k]\ \ do$
   $\quad receive\ \ packet_k^j,\ \ 1 \leq j \leq n\ \ and\ \ j \neq k$;

Figure 3: Data Movement: Approach 2

$receive[k]$ representing number of processors which will send packets to P_k. Finally, processors use this information to send and receive packets.

The complexity of this algorithm is difficult to analyze. The cost of steps 1 to 3 (Figure 3) is upper bounded by the parallel sum. The cost of step 4 and 5 in the worst case is difficult to analyze as it will depend on the network congestion and contention on which it is performed. A very loose upper bound on the complexity is $O(n^2(\tau + \varphi S))$. The performance of this algorithm should be much better in practice.

2.3 Approach 3

During the load balancing process, assume that P_k will left shift packets to P_i, where $k - maxl_k \leq i \leq k - minl_k$, $maxl_k$ and $minl_k$ represent P_k's maximum left shift distance and minimum left shift distance (> 0), respectively. These values can be calculated locally in $O(1)$ time. We observe that P_{k+1}'s maximum left shift distance $maxl_{k+1}$ must be less than or equal to $minl_k + 1$. With this observation, we know

that at any left shift stage, if P_k left shift packets to P_a and P_{k+1} left shift packets to P_b, then $a \leq b$. So we can conclude there is no link conflict at any shift stage. This is assuming that shift is carried over on an embedded linear array. The same conclusion holds for right shift operation.

The worst case time complexity of this algorithm (assuming that each node sends out a maximum of T packets to a maximum distance of D[1]) (Figure 4), is $O(T \cdot D(\tau + \varphi S))$. This is because each shift can be performed in $O(D(\tau + \varphi S))$ amount of time. This algorithm will be better than algorithm 1 and 2 if T and D are small.

2.4 Total Complexity

Thus the cost of load balancing is of the order to the cost of computing a parallel prefix followed by the time required for one of the approaches for data movement. The cost of parallel prefix is $O(\log n \cdot (\tau + \varphi))$ for hypercube architectures [17]. We believe that many of the future architectures would have some hardware support for such a primitive. In such case it can be assumed that parallel prefix can be calculated in $O(1)$ time; such is the case for CM-5 [20]. (Approach 1 has a better worst case time complexity than approach 2 and 3. However in practice, approach 2 and 3 may work better.)

Up to now, we have only performed the worst case complexity analysis. The worst case cost of the above algorithms makes them prohibitive for load balancing for many problems. However, as we shall show in the next section, the cost will be small if the granularity (amount of data) per node is reasonably large and the irregularity follows some reasonable distribution.

3 Probabilistic Analysis

We assume that each node has number of elements which are given by a distribution with mean μ and variance σ^2. We will derive results without any assumption on

[1] $D = \max\{Max_L_Shift, Max_R_Shift\}$

procedure Data_Movement();

1. $for \;\; i = maxl_k \;\; downto \;\; minl_k \;\; do$

   Perform a left shift of distance Max_L_Shift for $packet^k_{k-i}$ in a store and forward fashion. Whenever P_k receives a packet, if the packet is targeted to it, then P_k accepts this packet and removes it from communication channel. Otherwise, P_k forwards this packet toward its destination. If a node does not have any packet to send, it sends a dummy packet.

2. $for \;\; i = maxr_k \;\; downto \;\; minr_k \;\; do$

   Perform a right shift of distance Max_R_Shift for $packet^k_{k+i}$ in a store and forward fashion. Whenever P_k receives a packet, if the packet is targeted to it, then P_k accepts this packet and removes it from communication channel. Otherwise, P_k forwards this packet toward its destination. If a node does not have any packet to send, it sends a dummy packet.

Figure 4: Data Movement: Approach 3

the distribution and present specific results for normal distribution. Within the load balancing algorithm (Figure 1) there are two important parameters which typically affect the complexity of the algorithm,

Z : the maximum number of elements at any node. This will affect the maximum number of packets which are sent out by every node, and,

D : the maximum amount of distance which has to be traversed by a packet sent out by any node.

In the following analysis we study properties of the above two parameters. Towards this goal we first state a general result.

Let $U_1, \cdots, U_n$ be independent and identically distributed random variables with mean 0, variance 1, distribution function F, and associated density function f. Let

$$Z^* = \max\{U_1, \cdots, U_n\}.$$

Then, for large n, the distribution of normalized Z^* is given by the *extreme-value-distribution* [8]. More precisely,

$$\lim_{n\to\infty} P(b_n(Z^* - a_n) \leq x) = e^{-e^{-x}},$$

where a_n and b_n are sequences of constants satisfying

$$F(a_n) = \frac{n-1}{n}, \qquad b_n = n \cdot f(a_n).$$

From the properties of the extreme-value-distribution we know that

$$E(Z^*) = a_n + \frac{\gamma}{b_n}$$

where $\gamma = Euler's$ constant $= 0.5772$, and

$$Var(Z^*) = \frac{\pi^2}{6b_n^2}.$$

In particular, if U_i's are normally distributed, then both a_n and b_n are approximately equal to $\sqrt{2\ln n}$.

Now suppose that each X has the normal distribution function with mean μ and variance σ^2 and $Z = \max(X_1, \cdots, X_n)$. Then $Z = \mu + \sigma Z^*$ and substitution of mean and variance of Z^* gives

$$E(Z) = \mu + \sigma\left[\sqrt{2\ln n} + \frac{\gamma}{\sqrt{2\ln n}}\right],$$

and

$$Var(Z) = \frac{\pi^2\sigma^2}{6b_n^2} = \frac{\pi^2\sigma^2}{12\ln n}.$$

From the properties of the extreme value distribution described above we can evaluate

$$P\left[\frac{Z-\mu}{\sigma} \leq x\right] = e^{-e^{-(x-\sqrt{2\ln n})\sqrt{2\ln n}}}$$

for any x. For, $0 < \alpha < 1$, let

$$\alpha = e^{-e^{-(x-\sqrt{2\ln n})\sqrt{2\ln n}}},$$

then

$$x = \sqrt{2\ln n} + \frac{-\ln(-\ln\alpha)}{\sqrt{2\ln n}}.$$

So, in general the αth percentile of $(Z-\mu)/\sigma$ would be given by x and, for $n = 16,\ 64$, they are 3.6 and 3.9, respectively. It also means that for Z the αth percentile would be $\mu + \sigma x$, implying that $(Z-\mu)$ would have to go as much change as σx with probability $(1-\alpha)$. Consequently, probability that at least one processor will acquire a large number of elements is high even for small number of processors (if the variance is high).

In comparison with Z, distributional properties of D are considerably more involved. Let

$$V_k = X_1 + \cdots + X_k - k\overline{X}$$

where $\overline{X} = n^{-1}(X_1 + \cdots + X_n)$. Thus, $V_k/\overline{X}$ represents the amount of shift which is required for the first few elements of processor k. Distributional properties of V_k are easy to observe by rewriting

$$V_k = (1-\frac{k}{n})\,(X_1 + \cdots + X_k) - \frac{k}{n}\,(X_{k+1} + \cdots + X_n)$$

and recalling that each of the X's are independent random variables.

1. $E(V_k) = 0$
2. $Var(V_k) = \frac{k(n-k)}{n}\sigma^2$, $corr(V_k, V_l) = \sqrt{\frac{k(n-l)}{l(n-k)}}$, $k < l$
3. $V_n \equiv 0$
4. for $k = 1, \cdots, n-1$, distribution of V_k is given by the normal distribution $N(0, \frac{k(n-k)}{n}\sigma^2)$, if X's are normally distributed.

Thus behavior of each V_k is given by the properties of a normally distributed random variable. These properties of V_k's show that more deviation from zero will occur in the middle. Since V_k indicates amount of data movement from one processor to another, it would be useful to find probabilistic bounds on size of V_k's. For example, when $n = 16$, the eighth processor would encounter large data movement [variance of V_k is largest for $n = 16$] and since $P(|V_8|/\sigma > 4) = 0.05$ it follows that as much as $(4 \times \sigma)$ elements may have to move from this processor to some neighboring processors with probability 0.05. If $n = 64$, then as much as $(8 \times \sigma)$ elements may have to move from this processor in either direction with the same probability.

Now we consider properties of another random variable, W, which is of interest in analysis of D. This variable is defined as

$$W' = \frac{1}{\sigma\sqrt{n}}W = \max\{\frac{V_1}{\sigma\sqrt{n}}, \cdots, \frac{V_n}{\sigma\sqrt{n}}\}$$

Thus, random variable W represents maximum change among all processors. Properties of this random variable will allow us to quantify amount of data movement from one processor to others. Approximate asymptotic distribution of W' is obtained by realizing that the stochastic process generated by $V_1/\sigma\sqrt{n}$, $V_2/\sigma\sqrt{n}$, $\cdots$ is a Brownian Bridge. In other words, if we define

$$W^0(t) = \frac{V_{\lfloor nt \rfloor}}{\sqrt{n}\sigma} + (t - \frac{\lfloor nt \rfloor}{n})\frac{V_{\lfloor nt \rfloor + 1}}{\sqrt{n}\sigma}, \quad 0 \le t \le 1,$$

Then, as $n \to \infty$, the behavior of the process $\{W^0(t) : 0 \leq t \leq 1\}$ is such that (i) $E(W^0(t)) = 0$ for all t, (ii) $E(W^0(t)W^0(s)) = s(1-t)$ for $s \leq t$, and (iii) for all values of t the distribution of $E(W^0(t))$ is Gaussian.

Therefore, properties of this process can be used to obtain asymptotic distributions of interest. In particular, asymptotic distribution of W' is the same as the distribution of $\sup_{0 \leq t \leq 1} W^0(t)$ and the latter satisfies [3]:

$$P\left\{\sup_{0 \leq t \leq 1} W^0(t) \leq x\right\} = 1 - e^{-2x^2}, \quad x > 0.$$

Therefore, for large n

$$P(W' \leq x) = 1 - e^{-2x^2}, \quad x > 0.$$

In summary, the distribution of W, *i.e.*, $P(W \leq x)$, can be approximated by $1 - e^{-2(x^2/\sigma^2 n)}$ for $x > 0$. The αth percentile of W is easily obtained from this approximate distribution and is given by $\sigma\sqrt{n/2}(-\ln(1-\alpha))^{1/2}$. For example, when $\alpha = 0.95$ and $n = 16$, then the 0.95 percentile of (W/σ) is approximated by 4.895, and for $n = 64$ it goes up to 9.791. This is consistent with our previous observations about V's.

It would also be of interest to find the distribution of

$$D' = \frac{1}{\sigma\sqrt{n}} D^* = \max_{1 \leq k \leq n} \left\{ \left| \frac{V_1}{\sqrt{n}\sigma} \right|, \cdots, \left| \frac{V_k}{\sqrt{n}\sigma} \right|, \cdots, \left| \frac{V_n}{\sqrt{n}\sigma} \right| \right\}$$

which represents the maximum shift in either direction. However, our algorithms perform a shift along left followed by right. Hence the above distribution is not useful for evaluating the complexity of the algorithms. We give the following result for sake of completeness. Again using properties of the Brownian Bridge, we obtain the following asymptotic distribution for D': as $n \to \infty$, [3],

$$P(D' \leq x) \to P\left\{\sup_{0 \leq t \leq 1} \left|W^0(t)\right| \leq x\right\}$$

$$= 1 + 2\sum_{i=1}^{\infty}(-1)^i e^{-2i^2x^2}, \quad x > 0$$

Consequently, for large n, the distribution of D^*, $P(D^* \leq x)$, can be approximated by $1 + 2\sum_{i=1}^{\infty}(-1)^i e^{-(2i^2x^2/n\sigma^2)}$, for $x > 0$.

Returning back to W', it is easy to show that

$$E(W') = \frac{1}{2}\sqrt{\frac{\pi}{2}} = 0.626.$$

Finally, we consider the behavior of the normalized maximum right shift random variables

$$W^* = \max_{1 \leq k \leq n}\left\{\frac{V_1}{\overline{X}\sigma\sqrt{n}}, \cdots, \frac{V_n}{\overline{X}\sigma\sqrt{n}}\right\} = \frac{W}{\sqrt{n}\sigma\overline{X}} = \frac{W'}{\overline{X}} = \frac{D}{\sigma\sqrt{n}}\ .$$

By the strong law of large numbers, it follows that $\overline{X} \to \mu$ almost surely [19], and by Slutsky's Theorem [2], asymptotic distributions of W^* and D are *'essentially'* the same as of W'/μ and D'/μ respectively. Consequently, for large values of n, the following approximations can be used

$$P(W^* \leq x) = 1 - e^{-2x^2\mu^2}, \quad x > 0$$

(By symmetry, the distribution for maximum left shift should be similar.)

These distributions can be used to obtain desired probability bounds on the magnitudes of amount of data items sent from one processor to another.

From above, we have,

$$P(W^* \geq x) = e^{-2x^2\mu^2}, \quad x > 0$$

and

$$P(D \geq x) = e^{-\frac{2x^2\mu^2}{\sigma^2 n}}, \quad x > 0.$$

Now consider the expected time ξ to complete step 5 of load balancing algorithm, using the data movement algorithm in Approach 1. Realizing that $X \leq D\mu$ and using the property that it takes $O(D(\tau + X\varphi))$ time to move X amount of data, we get

$$\xi = \int_0^\infty (\lceil D \rceil \tau + \lceil D \rceil D\varphi\mu)\, f(D)dD$$

$$\leq \int_0^\infty \Big((D+1)\tau + (D^2 + D)\varphi\mu\Big)\, f(D)dD$$

$$= (E(D)+1)\tau + (E(D^2) + E(D))\varphi\mu \ .$$

Since $D = \max_{1 \leq i \leq n} |V_i|/\overline{X}$, therefore

$$\xi \leq (1 + 0.626\frac{\sigma\sqrt{n}}{\mu})\tau + (0.31\frac{\sigma^2 n}{\mu^2} + 0.626\frac{\sigma\sqrt{n}}{\mu})\varphi\mu \ .$$

The cost of left shift is also the same. Hence total cost of load balancing $= 2 \cdot \xi$.

The above gives the upper bound on the expected time for completion of our algorithm. In case $\mu \geq \sigma\sqrt{\frac{k}{2} n \ln n}$, we observe that

$$P(D \geq 1) = e^{-2\frac{(1)^2\mu^2}{\sigma^2 n}}$$

$$\leq e^{-\frac{2(1)^2\sigma^2 k n \ln n}{\sigma^2 n}}$$

$$= \frac{1}{n^k} \ .$$

Thus the probability of a shift of more than 1 unit in D is very low provided above property is satisfied by μ. This result indicates that most of the data movement occur among neighbor processors.

3.1 Discussion

From the analysis in the previous section, the cost of performing the data movement is

$$O(2(1 + 0.626\lambda)\tau + 2(0.31\lambda^2 + 0.626\lambda)\varphi\mu), \quad where\ \lambda = \frac{\sigma\sqrt{n}}{\mu}$$

Thus for all distribution with $\mu = O(\sigma\sqrt{n})$, the effective time for data shifting on an average is $O(\lambda(\tau + \varphi\mu))$. We will show in the next section that binomial distribution satisfies the above properties. Assuming that parallel prefix can be calculated reasonably efficiently (it can be calculated in $O(\tau \log n)$ for most architectures, and nearly constant time in architectures like CM-5), the cost of load balancing should make it practical for use for many applications. Further if τ is negligible when compared to $\varphi\mu$ and parallel prefix can be calculated in $O(1)$ time, then the total cost is proportioned to $O(\lambda\varphi\mu)$. Assuming that the cost of computation is at least proportional to number of elements in every local array, this result shows that the cost of load balancing should be no greater than the cost of computation. Typically load balancing needs to be performed after several iterations of computation. Our load balancing algorithms would add a small incremental cost if the above assumptions are satisfied.

4 A Simple Application

In the following we analyze the cost of load balancing for a specific instance. Assume that the input of a computational phase is a dense linear array which is distributed equally (each node has M elements). Assume that each element represents a computation with probability p (and no computation with a probability $1 - p$) which can be demonstrated by following statements (Figure 5).

The array A is distributed in a *block* distribution fashion so each processor has a local array $A[1..M]$. This would in general reduce the total communication. $C(M)$ represents the computation cost of the $if - then$ block. The cost in each node can be given by the binomial distribution $B(M, p)$. For reasonably large M this can be approximated by a normal distribution $N(\mu = Mp, \sigma^2 = Mp(1-p))$. Let $max_X = \max_{0 \le i < n} X_i$ (X_k represents the number of useful elements in P_k), the extra expected cost dues to load imbalance will be $C(M)(E(max_X) - \mu)$. If the cost is greater than the expected cost of load balancing (and possibly remapping), then it will benefit from the load balancing. Before proceeding further we make this comparison under the

```
for i = 1 to M * N do
    if condition (= TRUE with probability p) then
                    ⋮
        A[i] = f(A[i-1], A[i], A[i+1]);
                    ⋮
    endif
```

Figure 5: Simple application

assumption that $\frac{Mp}{1-p} > n$ and since, under our assumption $\lambda = \frac{\sigma\sqrt{n}}{\mu} = \sqrt{\frac{n(1-p)}{Mp}} < 1$, it follows that

$$C(M)(E(max_X) - \mu) \geq 2 \cdot \xi$$

$$\Rightarrow C(M)(E(max_X) - \mu) \geq 2[(1 + 0.626\lambda)\tau + (0.31\lambda^2 + 0.626\lambda)\varphi\mu]$$

It is observed that

$$C(M)(E(max_X) - \mu) \geq 2[(1 + 0.626\lambda)\tau + (0.31\lambda^2 + 0.626\lambda)\varphi\mu] \ .$$

We substitute the expected value of max_X for this case to obtain

$$C(M)\sigma\sqrt{2\ln n} \geq 2[(1 + 0.626\lambda)\tau + (0.31\lambda^2 + 0.626\lambda)\varphi\mu]$$

The above analysis has to be modified suitably if the cost of parallel prefix is not $O(1)$.

For example, for the CM-5 the time required for a scan operation is approximately 10 μsec, the value of τ is approximately 140 μsec, and the value of φ is approximately 0.5 $\mu sec/word$ (assuming a word size of 4 bytes). Assuming $M = 4096$, $n = 256$, and $p = 0.5$, we have

$$\mu = 2048, \quad \sigma = 32, \quad and \quad \lambda = 0.25 \ .$$

Neglecting the cost of parallel prefix, we have

$$C(M) \times 32 \times 3.33 \geq 2(1.156\tau + 360\varphi)$$

$$\Rightarrow C(M) \times 106.56 \geq 2.312\tau + 720\varphi$$

$$\Rightarrow C(M) \geq 0.022\tau + 6.756\varphi$$

Substituting $\tau = 140 \times 10^{-6} sec$ and $\varphi = 0.5 \times 10^{-6} sec$,

$$\Rightarrow C(M) \geq 6.458 \times 10^{-6}$$

Assuming a peak performance of 5 MFlops (the current CM-5 SPARC microprocessor), above analysis implies that we need approximately 30 instructions at right hand side. Thus load balancing will be preferable if the above condition is satisfied (which will be true for a large variety of applications). We should note that the value (in terms of number of instructions) of load balancing would go up if the processing speed increases (with the possible addition of vector units in CM-5).

5 Conclusions

In this paper, we present a simple load balancing algorithm and its probabilistic analysis. We demonstrate that the cost of load balancing is $O(\lambda(\tau + \varphi\mu))$ plus the cost of a parallel prefix. Our analysis indicate that in most practical cases the number of packets sent out by each processor is less than or equal to 2 (at most one on each side), and the size of these packets is almost surely less than or equal to the average number of elements on every node.

Our algorithms are suitable for most commercial architectures, which in most cases reduce the data movement to neighbor processors' shift operations. Our algorithms also preserve the data locality between data items which is extremely important in reducing inter-processor communication.

This paper provides load balancing only along one dimension. For many cases the data is distributed along two or more dimensions. We are currently analyzing a similar load balancing algorithms for two or more dimensions.

References

[1] Dana H. Ballard and Christopher M. Brown. *Computer Vision.* Prentice Hall, Englewood Cliffs, NJ, 1986.

[2] Bickel and Doksum. *Mathematical Statistics: Basic Ideas and Selected Topics.* Holden Day, 1977.

[3] Patrick Billingsley. *Convergence of Probability Measures*, page 85. John Wiley and Sons, NY, NY, 1968.

[4] T. Chou and J.A. Abraham. Load balancing in distributed systems. *IEEE Trans. on Software Engineering*, 8(4):401, July 1982.

[5] Alok Choudhary, Geoffrey C. Fox, Seema Hiranandani, Ken Kennedy, Charles Koelbel, Sanjay Ranka, and Chau-Wen Tseng. Compiling fortran 77d and 90d for mimd distributed-memory machines. In *Proceedings of the Frontiers of Massively Parallel Computation*, 1992. to appear.

[6] Y.C. Chow and W.H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Trans. on Computers*, 28(5):354, May 1979.

[7] Willian J. Dally and Chuck L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. on Computers*, 36(5):547, May 1987.

[8] H.A. David. *Order Statistics.* John Wiley and Sons, NY, NY, 1970.

[9] Geoffrey C. Fox. The architecture of problems and portable parallel software systems. Technical Report Revised SCCS-78b, Syracuse University, July 1991.

[10] E.K. Haddad. Partitioned load allocation for minimum parallel processing execution time. In *Proceedings of the 1989 International Conference on Parallel Processing*, page 192, 1989. Vol. II.

[11] D.Y. Hinz. A run time load balancing strategy for highly parallel systems. In *Proceedings of the 5th Distributed Memory Computing Conference*, page 951, Charleston, SC, April 1990.

[12] M.A. Iqbal, J.H. Saltz, and S.H. Bokhari. A comparative analysis of static and dynamic load balancing strategies. In *Proceedings of the 1986 International Conference on Parallel Processing*, page 1040, 1986.

[13] J. JáJá and K.W. Ryu. Load balancing and routing on the hypercube and related networks. Technical Report UMIACS-TR-89-61, CS-TR-2264, University Maryland, June 1989.

[14] L.M. Ni and K. Hwang. Optimal load balancing in a multiple processor system with many job classes. *IEEE Trans. on Software Engineering*, page 491, May 1985.

[15] D.M. Nicol and Jr. P.F. Reynolds. Optimal dynamic remapping of data parallel computations. *IEEE Trans. on Computers*, 39(2):206, February 1990.

[16] Michael J. Quinn. *Designing Efficient Algorithms for Parallel Computers.* McGraw-Hill, 1987.

[17] Sanjay Ranka and Sartaj Sahni. *Hypercube Algorithms with Applications to Image Processing and Pattern Recognition.* Springer-Verlag, 1990.

[18] V.A. Saletore. A distributed and adaptive dynamic load balancing scheme for parallel processing of medium-grain tasks. In *Proceedings of the 5th Distributed Memory Computing Conference*, page 994, Charleston, SC, April 1990.

[19] Galen R. Shorack and Jon A. Wellner. *Empirical Processes with Applications to Statistics.* John Wiley and Sons, NY, NY, 1986.

[20] Thinking Machines Corporation, Cambridge, MA. *The Connection Machine CM-5 Reference Manual*, 1992.