ELSEVIER Parallel Computing 25 (1999) 137–157

PARALLEL COMPUTING

Fundamental data movement operations and its applications on a hyper-bus broadcast network

Horng-Ren Tsai [a], Shi-Jinn Horng [b,*], Tzong-Wann Kao [c], Shung-Shing Lee [d], Shun-Shan Tsai [b]

[a] *Department of Information Management, The Overseas Chinese College of Commerce, Taichung, Taiwan, ROC*

[b] *Department of Electrical Engineering, National Taiwan University of Science and Technology, 43, Section 4, Kee-Lung Road, Taipei, Taiwan, ROC*

[c] *Department of Electronic Engineering, Kuang Wu Institute of Technology and Commerce, Taipei, Taiwan, ROC*

[d] *Department of Electronic Engineering, Fu Shin Institute of Technology and Commerce, I-Lain, Taiwan, ROC*

Received 16 April 1996; received in revised form 7 February 1998; accepted 7 July 1998

Abstract

A hyper-bus broadcast network (HBBN) consists of processors only sharing by some global buses, and there are no local links between processors. Based on such an architecture, we will exploit several efficient time parallel algorithms for solving the well-known fundamental data movement problems which had been extensively studied by researchers and widely applied to the field of image processing, digitized geometry and computer graphics. These include the leftmost one problem, the prefix maxima/minima problem, the m-contour problem, the all nearest neighbor problem and the all nearest smaller values problem, respectively. Note that the proposed algorithms not only can be implemented on the HBBN but also can be easily modified to run on other broadcast-based networks with the same time and processor complexities. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Prefix maxima/minima computation; m-contour problem; All nearest neighbor problem; All nearest smaller values problem; Image processing; Digitized geometry; Computer graphics; Parallel algorithm; Hyper-bus broadcast network

[*] Corresponding author. Tel: 886 2 27376700; fax: 886 2 27376699: e-mail: horng@mouse.ntust.edu.tw.

1. Introduction

Researchers have shown that the parallel random access machine (PRAM) models such as the exclusive read exclusive write (EREW) model, the CREW (concurrent read exclusive write) model and the CRCW (concurrent read concurrent write) model are too idealistic to be implementable currently. Even if parallel algorithms developed in these models are very efficient, they are only for theoretical interest. For example, consider the implementation of a CRCW PRAM model with p processors and a global memory of m words [16]. By definition, the processors are connected to the memory through a set of switching elements such that these p processors will be able to random access any distinct memory-cell of m words simultaneously. To ensure such connectivity, the total number of switching elements must be $\Theta(mp)$. For a reasonable memory size, it is very expensive to construct such a network. Thus, PRAM models of computation are impossible to realize in practice [16]. With the advance of VLSI technology, more practical and implementable parallel processing systems such as the array of processors, the mesh-connected computers, the hypercube multiprocessors and so on have been constructed by interconnection networks, even though the computation power of them is less that that of PRAM models [10,13]. A higher time complexity could be required for algorithms to be developed in such an architecture due to global communications but sometimes it can be developed as good as or even better than that developed in PRAM models. To overcome the long distance communications, embedding the existing parallel processing systems with global buses (usually called as broadcast-based networks) have been done by many researchers recently [1,4,7,9,12,14,19,20,22,23,30,33].

There are various features of the broadcast-based networks such as the broadcast communication model, the multi-channel broadcast network, the mesh-connected computers with multiple broadcasting, the generalized mesh-connected computers with multiple buses, the mesh-connected computers with hyperbus broadcasting and the reconfigurable network [1,4,7,9,12,14,19–24,30,32,33]. The broadcast communication model (BCM) [9,19,33] is the simplest architecture of the broadcast-based networks in which all processors are connected only by a global bus. Each processor in this model can communicate with others only through this broadcasting bus. For the sake of reducing the time complexity, the multi-channel broadcast network (MCBN) [21] is constructed as an enhanced model of the BCM by extending the number of broadcasting buses to k. For the sake of reducing the system diameter, the mesh-connected computers with multiple broadcasting (MCCMB) [4,17,31] are constructed by embedding the global buses to the mesh-connected computers. In order to increase the parallelization of the executing algorithm, the generalized mesh-connected computers with multiple buses (GMCCMB) [7] are proposed by partitioning MCCMB into several individual modules and all modules are connected through global buses only. Recently, Horng [12] extended the mesh-connected computers with hyperbus broadcasting (MCCHB). Such an extension led to this machine more powerful than other enhanced mesh-connected computers. Owing to the reconfigurability of the bus system, many researchers paid their attentions to the reconfigurable networks (RN) and had developed many constant time algorithms based on this architecture. An RN can be defined to be a set of processors connected to reconfigurable bus system whose configuration can be dynamically changed by properly setting local switches within each processor. Many RNs have been proposed including the reconfigurable meshes [23], the polymorphic torus architecture [20,22], the processor arrays with a reconfigurable bus system [32] and the reconfigurable array of processors [14].

A hyper-bus broadcast network (HBBN) is one of the broadcast-based networks. It consists of processors only sharing by some global buses and there are no local links between processors. Compared to the other broadcast-based networks, this architecture has the following two properties. First, by connecting processors with the global buses along each dimension constructed in the system, it can reduce the bus-contention problem caused by the BCM and the MCBN. Second, since it has no local buses between processors, it can save silicon area when it is directly implemented by a VLSI chip. In other words, it can be recognized as a reduced computation model of the MCCMB, GMCCMB, MCCHB and RN; obviously, the algorithms derived on the HBBN under the $O(\log N)$-bit natural bus width can be adapted to run on these models with the same time and processor complexities.

A well-known problem-solving paradigm that occurs in many computations is to reduce a large problem to one or several fundamental data movement problems and then solve these fundamental data movement problems efficiently. Thus, the more efficient fundamental data movement algorithms are exploited, the better performance of the executing algorithms based on them can be achieved. For example, the leftmost one problem, the prefix maxima/minima problem, the m-contour problem, the all nearest neighbor problem and the all nearest smaller values problem all are the well-known fundamental data movement problems, and they had been extensively studied by researchers and widely applied to the field of image processing, digitized geometry and computer graphics [2,3,6–8,11,15,17,25,26,29,31].

In this paper, we will develop several efficient parallel algorithms on the HBBN for solving the fundamental data movement problems. Note that the proposed algorithms not only can be implemented on the HBBN but also can be easily modified to run on the other broadcast-based networks such as the MCCMB, GMCCMB, MCCHB and RN with the same time and processor complexities.

The rest of this paper is organized as follows. We first describe the architecture upon out algorithms are based in Section 2. Section 3 discusses two fundamental data movement operations including the logical operation and the maximum/minimum computation. Section 4 develops several well-known fundamental data movement algorithms. Finally, some concluding remarks are included in the last section.

2. The hyper-bus broadcast network

In this section, we shall discuss the architecture of the HBBN which is the computation model adopted in this paper.

A 1-D HBBN (it can be recognized as a single-channel broadcast communication model) contains N processors. Each processor is identified by a 1-tuple unique index $i, 0 \leqslant i < N$. The processor with index i is denoted by P_i. Within each processor, there is one port S_1 connected to the global bus.

A 2-D HBBN contains $N_1 \times N_2$ processors as logically arranged in a 2-D grid. Each processor is identified by a 2-tuple unique index $(i,j), 0 \leqslant i < N_1, 0 \leqslant j < N_2$. The processor with index (i,j) is denoted by $P_{i,j}$. Within each processor, there are two ports, namely, S_1 and S_2, connected to the i-dimension and j-dimension global buses, respectively. We show an example for a 1-D HBBN of size 4 in Fig. 1(a) and also show another example for a 2-D HBBN of size 4×4 in Fig. 1(b).

Assume an HBBN is worked on a word model. For a unit of time, each processor can either perform arithmetic and logic operations or communicate with others by broadcasting data on a bus. It allows multiple processors to broadcast data on the different buses simultaneously at a time unit, if there is no collision. We allow multiple processors to broadcast data on the different buses or to broadcast the same data on the same bus simultaneously at a time unit. If more than one processor attempts to broadcast different data on the same bus simultaneously, then a collision occurs and the final data received are unexpected. That is, each processor can operate on an arbitrarily individual bit of the global bus simultaneously. This assumption has been used to solve various problems by many researchers [14,30]. Practically, the concurrent write ability is implemented in the content-addressable array parallel processor as proposed by Shu et al. [30].

One may criticize that the bus driving capacity is limited and there is a propagation delay for the bus. Technically, the number of fan-out of a CMOS driver is limited. This problem can be improved by either increasing the driving power or through the bus buffer. The propagation delay is dependent on the interconnect material. Metal interconnects will result larger delay than that of using optical fibers. For example, if the length of the optical fiber is one meter then the propagation delay along this optical fiber is about 2/3 ns. See Ref. [28], for details. Currently, it seems quite impossible to design a chip of size 1 m by 1 m. Hence, for a practical size of the HBBN, both problems caused by the bus are negligible.

An HBBN is operated in an single instruction stream, multiple data streams (SIMD) model. An *enable/disable mask* can be used to select a subset of the processing elements that are to perform an instruction. Only the enabled processors will perform the same instruction. The remaining processors will be idle. As for easily presenting out algorithms on a 1-D HBBN or a 2-D HBBN, let var(i) (or var(i,j)) denote the local variable var (memory or register) in a processor with index (i) (or (i,j)). For example, sum(0, 1) is a local variable sum of processor $P_{0,1}$. The complexity of an algorithm is assumed to be the sum of the maximal computation time among all processors and the communication time among all processors and this assumption was also used by many researchers [1,3,4,6,9,11,14,17,19–23,26,25,30–33].

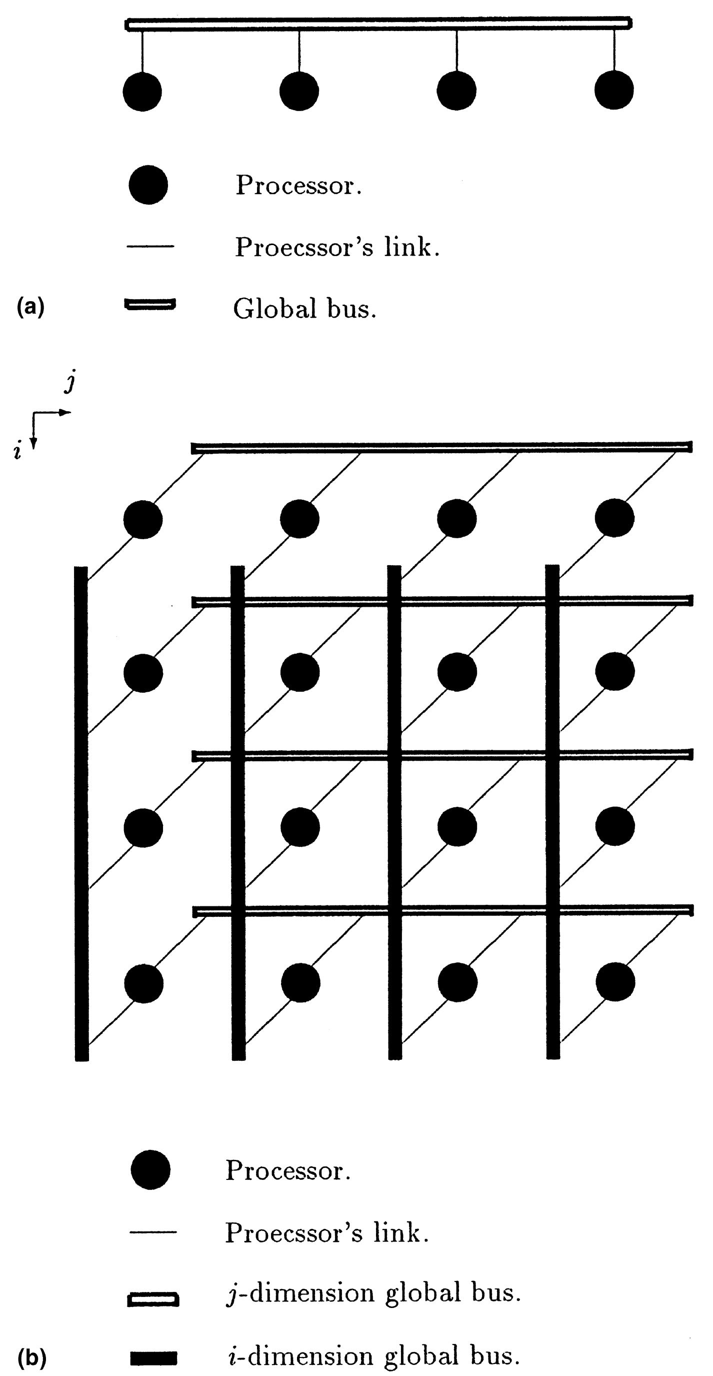


Fig. 1. (a) A 1-D HBBN of size 4. (b) A 2-D HBBN of size 4 × 4.

3. Fundamental data movement operations

Two data operations will be described in this section. These two data operations are used for developing several efficient fundamental data movement algorithms in the following sections.

3.1. The logical operation

Given a linear HBBN of size N, in which each processor P_i, $0 \leqslant i < N$, stores a Boolean data, the problem is to determine the logical OR (logical AND) of these N Boolean data. By the concurrent-written ability of the global bus, the logical OR (logical AND) of N Boolean data can be determined by the following two major steps. First, processor $P_i, 0 \leqslant i < N$, writes a signal 1 to the global bus, if it holds a true (false) data. Then, processor P_0 reads the data from the global bus. If processor P_0 reads a signal 1 from the global bus, then the logical OR (logical AND) of N Boolean data is true (false); otherwise, it is false (true). Obviously, the time complexity of two major steps is O(1). Hence, this leads to the following lemma.

Lemma 1. *Given a Boolean data sequence of size N, each processor holds a Boolean data, the logical* OR/AND *of these N Boolean data can be determined in* O(1) *time on a* 1-D HBBN *using N processors.*

3.2. The maximum/minimum computation

Given N unsigned integers $A_0, A_1, \ldots, A_{N-1}$ each of size $\log N$-bit, where $0 \leqslant A_i < N$ and $0 \leqslant i < N$, the maximum (minimum) computation of these N integers is to find an integer which is greater (less) than the others. On the CRCW PRAM model, two algorithms, one runs in O(1) time using N^2 processors and the other runs in $O(\log\log N)$ time using N processors, for this problem have been well stated by Chaudhuri [5]. On the broadcast-based network, Dechter and Kleinrock [9] gave an $O(\log N)$ time algorithm for this problem on the BCM using N processors by the binary tree technique. Miller et al. [23] also gave two parallel algorithms for this problem on the reconfigurable mesh: one runs in $O(\log N)$ time on a 1-D reconfigurable mesh using N processors and the other runs in $O(\log\log N)$ time on a 2-D reconfigurable mesh using $N^{1/2} \times N^{1/2}$ processors. For improving the 1-D result proposed by Miller et al. [23], Kao and Horng [14] also gave an $O(T)$ time algorithm for this problem on a 1-D reconfigurable array of processors using N processors, where the bus width is w-bit for $\log N \leqslant w < N$ and $T = \lfloor \log_w N \rfloor + 1$. This algorithm is quite efficient and easily to be implemented on the other broadcast-based network. For the sake of completeness, their main idea is summarized in the following.

Without loss of generality, assume that all numbers are distinct and each A_i is represented by the base-2 number system as follows:

$$A_i = \sum_{k=0}^{B-1} b_{i,k} 2^k, \tag{1}$$

where $B = \lfloor \log N \rfloor + 1$, $b_{i,k} \in \{0, 1\}$, $0 \leqslant A_i < N$, and $0 \leqslant i < N$.

Instead of using the base-2 number system, A_i can be also represented by the base-w number system as follows

$$A_i = \sum_{k=0}^{T-1} a_{i,k} w^k, \tag{2}$$

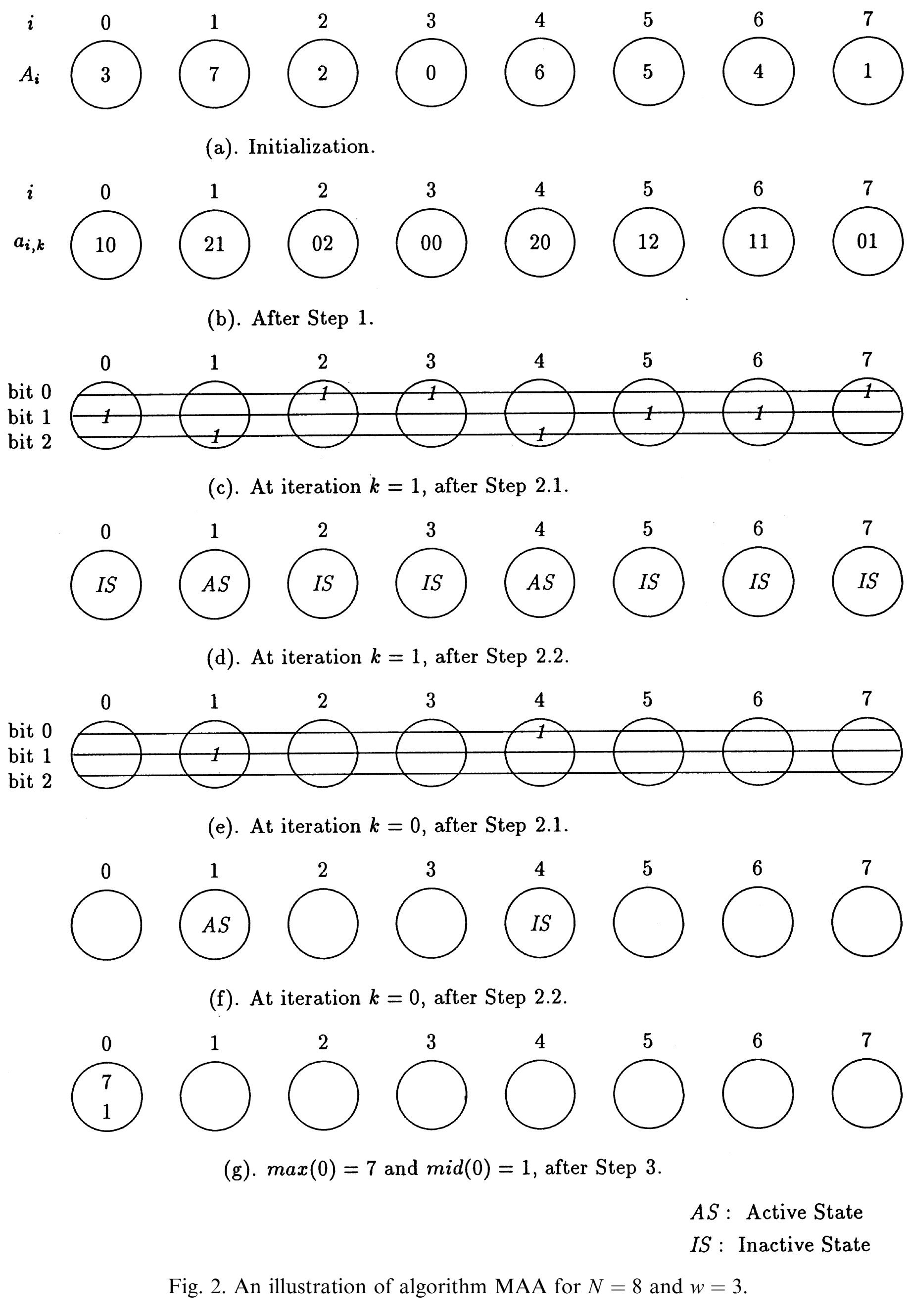
where $T = \lfloor \log_w N \rfloor + 1$, $0 \leqslant i < N$ and $0 \leqslant a_{i,k} < w$.

Based on Eq. (2), the maximum (minimum) number of these N unsigned integers can be found by the following two major steps. In the first major step, apply the division operation on A_i to obtain $a_{i,k}$ for $0 \leqslant k \leqslant T-1$. By this way, each A_i is represented by T digits and each digit is bounded within the interval $[0, w-1]$. In the second step, apply the prune-and-search technique to find the maximum (minimum) number of these N unsigned integers. If $a_{i,k}$ of A_i is greater (less) than $a_{j,k}$ of A_j then A_j could be pruned by A_i. Continuing this process from the most significant digit to the least significant digit, the maximum (minimum) number can be found at most T iterations. Let the global bus width be w-bit. Initially, the state of each processor is set in "active" and the global bus is cleared to 0. Each A_j is stored in processor P_j, $0 \leqslant j < N$. Finally, the maximum (minimum) number and its associated index are stored in the local variables max(0) and mid(0) of processor P_0, respectively. We show the detailed maximum algorithm (MAA) in the following. The minimum algorithm can be derived similarly. Assume $N = 8$ and $w = 3$. Fig. 2 shows an example for the data 3, 7, 2, 0, 6, 5, 4 and 1 to be executed by algorithm MAA.

Algorithm MAA (A, max, mid);

```
/* A is an input variable. max and mid are output variables. */
Step 0. begin.
Step 1. Processor $P_i$, $0 \leqslant i < N$, computes $a_{i,k}$ from $A_i$ by Eq. (2) for $0 \leqslant k < T$.
Step 2. repeat Steps 2.1 and 2.2 from $k = T-1$ to 0.
    Step 2.1. Processor $P_i, 0 \leqslant i < N$, writes a signal 1 to the bit $r$ of the global bus,
    if it is in state "active" and $a_{i,k} = r$, for $0 \leqslant r < w$.
    Step 2.2. Processor $P_i$ with $a_{i,k} = r$ for $0 \leqslant i < N$ and $0 \leqslant r < w$, sets its state "ac-
    tive", if the state of $P_i$ is "active" and the bits $r+1, r+2, \ldots, w-1$ read from
    the global bus all are 0; otherwise, sets its state "inactive".
Step 3. Processor $P_i$, $0 \leqslant i < N$, whose state is in "active", copies $A_i$ and index $i$ to
max(0) and mid(0) of processor $P_0$ using the global bus, respectively.
Step 4. end.
```

The time complexity of algorithm MAA is analyzed as follows. Steps 1 and 2 require $O(T)$ time and Step 3 requires $O(\lceil \log N/w \rceil)$ time as the bus width is w-bit. Hence, the time complexity is $O(T + \lceil \log N/w \rceil)$.



i
0 1 2 3 4 5 6 7
A_i
3 7 2 0 6 5 4 1
(a). Initialization.
$a_{i,k}$
10 21 02 00 20 12 11 01
(b). After Step 1.
bit 0
bit 1
bit 2
(c). At iteration $k = 1$, after Step 2.1.
IS AS IS IS AS IS IS IS
(d). At iteration $k = 1$, after Step 2.2.
(e). At iteration $k = 0$, after Step 2.1.
AS IS
(f). At iteration $k = 0$, after Step 2.2.
7
1
(g). $max(0) = 7$ and $mid(0) = 1$, after Step 3.
AS : Active State
IS : Inactive State

Fig. 2. An illustration of algorithm MAA for $N = 8$ and $w = 3$.

Lemma 2. *Algorithm* **MAA** *can be computed in* $O(T + \lceil \log N/w \rceil)$ *time on a* 1-D HBBN *using N processors, where the bus width of the global bus is w-bit and* $T = \lfloor \log_w N \rfloor + 1$.

For the nondistinct data, the signed integer data and the real number data, the maximum (minimum) of each can be easily obtained by modifying the algorithm MAA, respectively. Hence, the following lemmas as stated in Ref. [14] is also true in the HBBN.

Lemma 3. *Given N real numbers each of size* $O(\log N)$*-bit the mantissa and exponent parts each takes* $\log N$*-bit, the maximum/minimum of these N numbers can be found in* $O(T + \lceil \log N/w \rceil)$ *time on a 1-D* HBBN *using N processors, where the bus width of the global bus is w-bit for* $2 \leqslant w < N$ *and* $T = \lfloor \log_w N \rfloor + 1$.

Based on Lemma 3, algorithm MAA can be run in $O(\log N/\log\log N)$ time under the $O(\log N)$-bit natural bus width. It is quite efficient even if w is less than $\log N$. Assume there are 2^{32} processors and each has a data of size 32-bit. Also assume the bus width of each processor is 16-bit. Then, it requires only O(11) unit time to find the maximum (minimum) of these 2^{32} data. In particular, if the bus width is extended to $N^{1/c}$-bit (for $N^{1/c} > \log N$), where c is a constant and $c \geqslant 1$. Then, $T + \lceil \log N/w \rceil = \lfloor \log_{N^{1/c}} N \rfloor + 1 + \lceil \frac{\log N}{N^{1/c}} \rceil = \lfloor c \rfloor + 2$, is also a constant.

To justify the hardware cost of a 2-D $N \times N$ HBBN, a practical measuring result of the VLSI chip of the reconfigurable mesh, called the Yorktown Ultra Parallel Polymorphic Image Engine (YUPPIE) implemented by Maresca and Li [20,22], could be used. The YUPPIE VLSI chip was fabricated with 2μ CMOS technolgy. It contained 16 processors arranged in a 4×4 mesh each with the switch function. Each processor was equipped with a one-bit ALU, 5 registers and 256 bit memory. The final chip size was 5.0×6.5 mm (excluding I/O pads) and the chip had 68 pins. The chip area was divided into three main blocks: 24% for the memory block, 38% for the processor block and 12% for the switch function and mesh wires. According to the design, the switch function and mesh wires took roughly one fifth of the processor and the 256-bit memory silicon area (i.e., 12:62). The ratio indicated that the hardware cost of the switch function and mesh wires were fairly low. For example, assume that there are 2^{32} processors and each has a data of size 32-bit. Also assume the bus width of each processor is 32-bit. Let $c = 6$ then $N^{1/c} = 2^{32/6} = 2^{5.33} = 40.32$. That is, when the bus width w is extended to 41-bit, it requires only O(8) unit time to find the maximum (minimum) of these 2^{32} data items. The wiring ratio between $N^{1/c}$-bit and $\log N$-bit for the above example would be

$$\frac{N^{2+2/c}}{N^2 \log^2 N} = \frac{2/c}{\log^2 N} = \frac{2^{64/6}}{2^10} = 2^{4/6} = 1.58.$$

Since the processor and memory sizes are increased with the extended bus width, the chip area by the wiring is increased far less than from 20% to 31.6% relatively to the extended processor and memory. The wider bus of the system installed, the more powerful parallel processing system created. As stated before, the area occupied by the extended bus relatively to the newly created chip is limited. Therefore, adding some extra buses to each processor on a 2-D HBBN could not increase the hardware cost of the system enormously.

Corollary 1. *Given N real numbers each of size* $O(\log N)$*-bit (the mantissa and exponent parts each takes* $\log N$*-bit), the maximum (minimum) of these N data can be computed in* $O(1)$ *time on a* 1-D HBBN *using N processors, where the bus width of the global bus is* $N^{1/c}$*-bit* ($N^{1/c} > \log N$) *for a constant c and* $c \geqslant 1$.

Note that some researchers criticized there was a local switch delay for the reconfigurable bus of any RNs. Hence, the time complexity of the algorithms derived in this paper is more efficient than that derived in Ref. [14] if the local switch is included.

4. Applications

In this section, we will develop several parallel algorithms for solving the well-known fundamental data movement problems. These include the leftmost one problem, the prefix maxima/minima problem, the *m* contour problem, the all nearest neighbor problem and the all nearest smaller values problem, respectively.

4.1. The leftmost one problem

Given a Boolean of size $N \times N$ with every element of it being either 0 or 1, the *leftmost one problem* is to determine the position of the leftmost 1 of each row of the matrix. The leftmost one problem is a useful operation and has some applications in image processing, digitized geometry and computer graphics [11]. Finding the leftmost one on meshes with row broadcasting, Stout [31] first proposed an $O(N^{1/6})$ time algorithm for this problem using a 2-D $N \times N$ processors. Their algorithm [31] ran under the assumption that all processors had an unbounded local memory. Recently, instead of using unbounded local memory of each processor, Gurla [11] achieved the same time complexity as that derived in Ref. [31] but the memory used in each processor was reduced to a constant.

This problem can be solved in O(1) time on a 2-D $N \times N$ HBBN. Initially, these $N \times N$ Boolean data are stored in processor $P_{i,j}$, $0 \leqslant i, j < N$. Finally, the position of the leftmost 1 of each row of this matrix is stored in the local variable $l(i,0)$ of processor $P_{i,0}$, $0 \leqslant i < N$. Our algorithm consists of the following two major steps. First, apply Lemma 1 to identify whether there is an element 1 or not in each row of 2-D HBBN. Then, for each row i, $0 \leqslant i < N$, if it does not exist any element 1 then processor $P_{i,0}$ sets $l(i,0)$ to nil. Otherwise, it sets $l(i,0)$ to the position of the leftmost one (the minimum index of all element 1's) found in row i by Corollary 1. These two steps take O(1) time by Lemma 1 and Corollary 1, respectively. This leads to the following theorem.

Theorem 1. *Given a Boolean matrix of size $N \times N$, the position of the leftmost one of each row of this matrix can be found in* $O(1)$ *time on a* 2-D HBBN *using $N \times N$ processors, where the bus width of the global bus is $N^{1/c}$-bit ($N^{1/c} > \log N$) for a constant c and $c \geqslant 1$.*

4.2. The prefix maxima/minima problem

Given a data sequence of N numbers $A_0, A_1, \ldots, A_{N-1}$ each of size $O(\log N)$-bit, the prefix maxima problem is to compute $PA_j = \max\{A_0, A_1, \ldots, A_j\}$ for all j, where $0 \leqslant j < N$. The prefix maxima/minima operation is a fundamental tool in designing efficient parallel algorithms for many problems.

This problem can be solved in O(1) time on a 2-D $N \times N$ HBBN. Initially, assume that the data A_j are stored in processor $P_{0,j}$, $0 \leqslant j < N$. Finally, the prefix maxima of $A_0, A_1, \ldots, A_j$ is stored in the local variable $PA(0, j)$ of processor $P_{0,j}$, $0 \leqslant j < N$. Our algorithm consists of the following three major steps. Step 1, processor $P_{0,j}$, $0 \leqslant j < N$, copies A_j to $P_{i,j}$ through the i-dimension global bus for $0 \leqslant i < N$. Step 2, processor $P_{i,j}$ sets its state in active if $0 \leqslant j \leqslant i$, otherwise, sets its state in inactive. Then, for each row i, $0 \leqslant i < N$, processor $P_{i,j}$, $0 \leqslant j < N$, applies Corollary 1 to compute the maximum on $A_0, A_1, \ldots, A_i$ through the j-dimension global bus, and let the computed result be stored in the local variable $PA(i, 0)$ (i.e., $PA(i, 0) = \max\{A_0, A_1 \ldots A_i\}$). Step 3, processor $P_{i,0}, 0 \leqslant i < N$, copies $PA(i, 0)$, $0 \leqslant i < N$, to $PA(i, i)$ through the j-dimension global bus; and then processor $P_{j,j}$, $0 \leqslant j < N$, copies $PA(j, j)$ back to $PA(0, j)$ through the i-dimension global bus. Since Steps 1 and 3 take O(1) time and Step 2 takes O(1) time by Corollary 1, the total time complexity is O(1). Hence, this leads to the following theroem.

Theorem 2. *Given a data sequence of N real numbers each of size* $O(\log N)$*-bit, the prefix maxima of these N numbers can be computed in* $O(1)$ *time on a* 2-D HBBN *using $N \times N$ processors, where the bus width of the global bus is $N^{1/c}$-bit ($N^{1/c} > \log N$) for a constant c and $c \geqslant 1$.*

Similarly, the prefix minima and postfix maxima/minima of N real numbers each of size $O(\log N)$-bit can be also computed in O(1) time on a 2-D HBBN using N^2 processors, respectively.

4.3. The m-contour problem

The *m-contour* problem is defined as follows. Given a set of N arbitrary points $v_0, v_1, \ldots, v_{N-1}$ in the Euclidean plane, and each v_i is specified by its cartesian coordinates (x_i, y_i), the m-contour problem is to determine these maximal points under the Euclidean plane. That is, for each point v_i, it is not a contour point under Euclidean plane if there exists a point v_j such that $x_i < x_j$ and $y_i < y_j$; otherwise, v_i is a contour point. The m-contour problem is a basic problem is computational geometry [27].

The main idea of our algorithm for solving the *m*-contour problem is described as follows. For each point $v_i, 0 \leqslant i < N$, we can use N processors to compare the cartesian coordinates of it and those of other $N - 1$ points. Then, the contour point of each point v_i can be determined by checking whether there does not exist any point whose cartesian coordinates are greater than those of v_i. This problem can be solved in O(1) time on a 2-D $N \times N$ HBBN. Initially, assume these N points are stored in processor $P_{0,j}$, $0 \leqslant j < N$. Finally, a flag $c(0,j)$ is stored in processor $P_{0,j}$, $0 \leqslant j < N$. $c(0,j) = 1$ if point v_j is a contour point; $c(0,j) = 0$, otherwise. Our algorithm consists of the following four major steps. Assume $v_0 = (2,0)$, $v_1 = (1,3)$, $v_2 = (3,1)$ and $v_3 = (0,2)$. A snapshot for solving the *m*-contour problem is also shown in Fig. 3. Step 1, processor $P_{0,j}$, $0 \leqslant j < N$, copies the cartesian coordinates (x_j, y_j) of v_j to $P_{i,j}$, $0 \leqslant i < N$, through the *i*-dimension global bus; and then processor $P_{i,i}$, $0 \leqslant i < N$, copies the cartesian coordinates (x_i, y_i) for v_i to $P_{i,j}$, $0 \leqslant j < N$, through the *j*-dimension global bus. Thus, processor $P_{i,j}$ holds the cartesian coordinates (x_i, y_i) and (x_j, y_j) of points v_i and v_j, respectively. Step 2, processor $P_{i,j}$, $0 \leqslant i,j < N$, sets $c(i,j) = 0$, if $x_i < x_j$ and $y_i < y_j$; sets $c(i,j) = 1$, otherwise. Step 3, for each row $i, 0 \leqslant i < N$, processor $P_{i,j}, 0 \leqslant j < N$, applies Lemma 1 on $c(i,j)$ to determine the contour condition through the *j*-dimension global bus, and the result is store in $c(i,0)$. Step 4, processor $P_{i,0}$, $0 \leqslant i < N$, copies $c(i,0)$ to $c(i,i)$ through the *j*-dimension global bus; and then processor $P_{j,j}, 0 \leqslant j < N$, copies $c(j,j)$ back to $c(0,j)$ through the *i*-dimension global bus. Steps 1, 2 and 4 take O(1) time, respectively. Step 3 takes O(1) time by Lemma 1. Hence, the total time complexity is O(1). This leads to the following theorem.

Theorem 3. *Given a set of N arbitrary points* $v_0, v_1, \ldots, v_{N-1}$ *in the Euclidean plane, the m-contour problem can be solved in* O(1) *time on a* 2-D HBBN *using* $N \times N$ *processors, where the bus width of the global bus is* $N^{1/c}$*-bit* ($N^{1/c} > \log N$) *for a constant c and* $c \geqslant 1$.

4.4. The all nearest neighbor problem

The *all nearest neighbor* (ANN) problem is defined as follows. Given a set of N arbitray points $v_0, v_1, \ldots, v_{N-1}$ in the Euclidean plane, this problem is to determine the nearest neighbor for each point. In other words, for each point $v_i, 0 \leqslant i < N$, find the nearest neighbor point $v_j, j \neq i, 0 \leqslant j < N$. For measuring the distance, the Euclidean distance (L_2 metric) is considered. It can be also extended to operate on any L_k metric. The distance between any two points (x_i, y_i) and (x_j, y_j) is defined by

$$nb(i,j) = \{(x_i - x_j)^2 + (y_i - y_j)^2\}^{1/2}, \tag{3}$$

where $0 \leqslant i,j < N$ and $i \neq j$.

The ANN problem is a basic problem in computational geometry, and it has a number of applications. See Ref. [27], for details. It also can be referred to as "proximity" or "closeness" being fundamental in image processing, pattern

i \ j	0	1	2	3
1	(2,0)	(1,3)	(3,1)	(0,2)
1				
2				
3				

(a)

i \ j	0	1	2	3
1	(2,0) (2,0)	(1,3) (2,0)	(3,1) (2,0)	(0,2) (2,0)
1	(2,0) (1,3)	(1,3) (1,3)	(3,1) (1,3)	(0,2) (1,3)
2	(2,0) (3,1)	(1,3) (3,1)	(3,1) (3,1)	(0,2) (3,1)
3	(2,0) (0,2)	(1,3) (0,2)	(3,1) (0,2)	(0,2) (0,2)

(b)

	0	1	2	3
1	1	1	0	1
1	1	1	1	1
2	1	1	1	1
3	1	0	1	1

(c)

	0	1	2	3
1	0			
1	1			
2	1			
3	0			

(d)

	0	1	2	3
1	0	1	1	0
1				
2				
3				

(e)

Fig. 3. An example for solving the m-contour problem. (a) Initalization; (b) After Step 1; (c) After Step 2; (d) After Step 3; (e) After Step 4.

recognition and computer graphics [26]. The ANN problem has been extensively studied in both sequential and parallel algorithms [8,18,26,27,29]. Lee and Preparata [18] proposed an $O(N \log N)$ time sequential algorithm for this problem. Cole and Goodrich [8] proposed an $O(\log N)$ time parallel algorithm for this problem on the CREW PRAM model using $O(N)$ processors. Under the assumption that all points were given in counter-clockwise order, Schieber and Vishkin [29] proposed two parallel algorithms for this problem on the CRCW PRAM model; one ran in $O(\log \log N)$ time using $N/\log \log N$ processors, the other ran in $O(1)$ time using $N^{2+\epsilon}$ processors for a constant ϵ, respectively. Recently, Olariu and Stojmenovic [26] proposed an $O(\log N)$ time algorithm for this problem on a 2-D MCCMB using $N \times N$ processors.

The main idea of our algorithm for solving the ANN problem can be described as follows. For each point v_i, $0 \leqslant i < N$, we can use N processors to compute the Euclidean distance between v_i and each of other $N-1$ points. Then, the nearest neighbor of each point v_i can be found by computing the minimum distance of these $N-1$ Euclidean distances.

This problem can be solved in $O(1)$ time on a 2-D $N \times N$ HBBN. Initially, assume that these N points are stored in processor $P_{0,j}$, $0 \leqslant j < N$. Finally, the distance and the index of the nearest neighbor point of each point v_j are stored in the local variables $nb(0,j)$ and $nbi(0,j)$ of processor $P_{0,j}$, respectively. Our algorithm consists of the following four major steps. Assume $v_0 = (2,0)$, $v_1 = (1,3)$, $v_2 = (3,1)$ and $v_3 = (0,2)$. A snapshot for solving the ANN problem is shown in Fig. 4. Step 1, processor $P_{0,j}$, $0 \leqslant j < N$, copies the cartesian coordinates (x_j, y_j) of v_j to $P_{i,j}$, $0 \leqslant i < N$, through the i-dimension global bus; and then processor $P_{i,i}$, $0 \leqslant i < N$, copies the cartesian coordinates (x_i, y_i) of v_i to $P_{i,j}$, $0 \leqslant j < N$, through the j-dimension global bus. Thus, each processor $P_{i,j}$ holds the cartesian coordinates (x_i, y_i) and (x_j, y_j) of points v_i and v_j, respectively. Step 2, processor $P_{i,j}$, $0 \leqslant i$, $j < N$ and $i \neq j$, computes the Euclidean distance of v_i and v_j by Eq. (3), and stores the result in $nb(i,j)$. Then, processor $P_{i,i}$ sets $nb(i,i)$ to ∞ for $0 \leqslant i < N$. Step 3, for each row i, $0 \leqslant i < N$, processor $P_{i,j}$, $0 \leqslant j < N$, applies Corollary 1 on $nb(i,j)$ to find the minimum Euclidean distance through the j-dimension global bus; and let $nb(i,0)$ and $nbi(i,0)$ be used to store the minimum distance and the index of the nearest neighbor point, respectively. Step 4, processor $P_{i,0}$, $0 \leqslant i < N$, copies $nb(i,0)$ and $nbi(i,0)$ to $nb(i,i)$ and $nbi(i,i)$ through the j-dimension global bus, respectively; and then processor $P_{j,j}, 0 \leqslant j < N$, copies $nb(j,j)$ and $nbi(j,j)$to $nb(0,j)$ through the i-dimension global bus, respectively. The time complexity is analyzed as follows. Step 1, 2 and 4 take $O(1)$ time, respectively. Step 3 takes $O(1)$ time by Corollary 1. Hence, the total time complexity is $O(1)$ time. This leads to the following theorem.

Theorem 4. *Given a set of N arbitrary points $v_0, v_1, \ldots, v_{N-1}$ in the Euclidean plane, the ANN problem can be solved in* $O(1)$ *time on a* 2-D HBBN *using $N \times N$ processors, where the bus width of the global bus is $N^{1/c}$-bit* $(N^{1/c} > \log N)$ *for a constant c and* $c \geqslant 1$.

j

i	0	1	2	3
1	(2, 0)	(1, 3)	(3, 1)	(0, 2)
1				
2				
3				

(a)

j

i	0	1	2	3
1	(2, 0) (2, 0)	(1, 3) (2, 0)	(3, 1) (2, 0)	(0, 2) (2, 0)
1	(2, 0) (1, 3)	(1, 3) (1, 3)	(3, 1) (1, 3)	(0, 2) (1, 3)
2	(2, 0) (3, 1)	(1, 3) (3, 1)	(3, 1) (3, 1)	(0, 2) (3, 1)
3	(2, 0) (0, 2)	(1, 3) (0, 2)	(3, 1) (0, 2)	(0, 2) (0, 2)

(b)

	0	1	2	3
1	∞	$\sqrt{10}$	$\sqrt{2}$	$\sqrt{8}$
1	$\sqrt{10}$	∞	$\sqrt{8}$	$\sqrt{2}$
2	$\sqrt{2}$	$\sqrt{8}$	∞	$\sqrt{10}$
3	$\sqrt{8}$	$\sqrt{2}$	$\sqrt{10}$	∞

(c)

	0	1	2	3
1	$\sqrt{2}$ 2			
1	$\sqrt{2}$ 3			
2	$\sqrt{2}$ 0			
3	$\sqrt{2}$ 1			

(d)

	0	1	2	3
1	$\sqrt{2}$ 2	$\sqrt{2}$ 3	$\sqrt{2}$ 0	$\sqrt{2}$ 1
1				
2				
3				

(e)

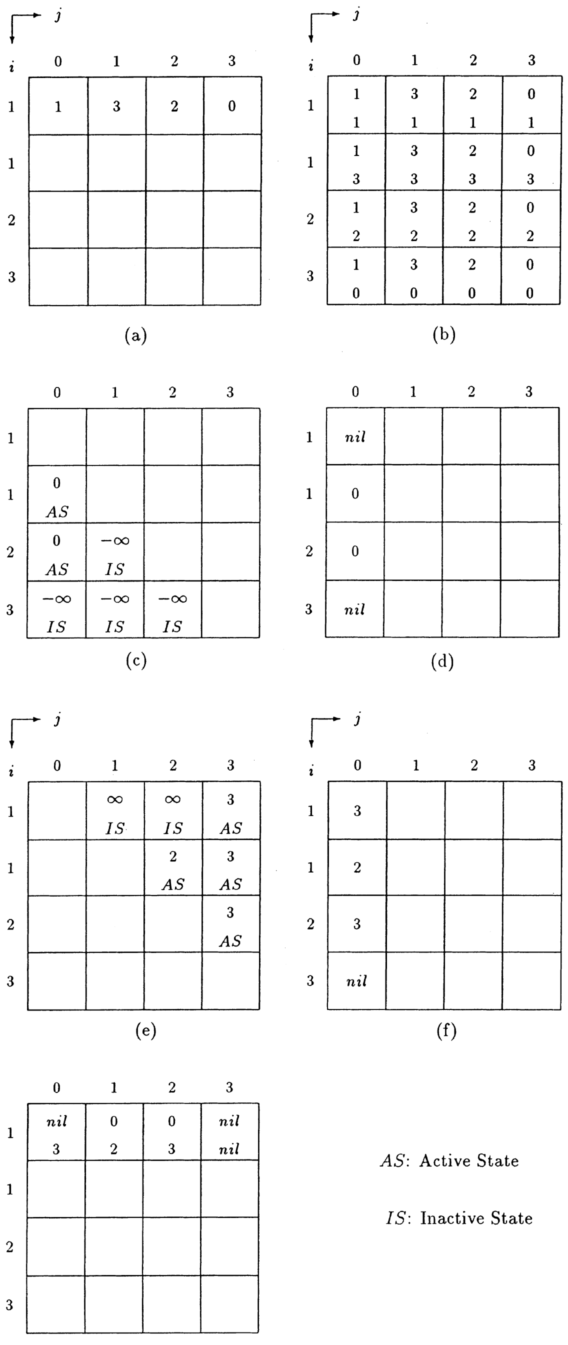
Fig. 4. An example for solving the ANN problem. (a) Initalization; (b) After Step 1; (c) The nb(i, 0), after Step 2; (d) The nb(i, 0) and nbi(i, 0) and nbi(i, 0) after Step 4; (e) The nb(0, j) and nbi(0, j), after Step 4.

4.5. The all nearest smaller values problem

The *all nearest smaller values* (ANSV) problem is defined by Berkman et al. [2]. The authors also argue that this is a fundamental problem on parallel processing, and many other problems can be reduced to it. The ANSV problem can be formulated as follows. Given a data sequence $a_0, a_1, \ldots, a_{N-1}$ of N elements from a totally ordered domain, the problem is to determine each element a_i whose nearest element to its left and nearest element to its right are both smaller than a_i. In other words, for each i, $0 \leqslant i < N$, find the maximal j, $0 \leqslant j < i$ such that $a_j < a_i$, and find the minimal k, $i < k < N$ such that $a_k < a_i$. We say that a_j and a_k, if exist, are the *left match* (denote by lm(i)) and *right match* (denoted by rm(i)) of a_i, respectively; otherwise, let lm(i) = nil or rm(i) = nil.

For solving the ANSV problem, Berkman et al. [2] proposed an $\mathrm{O}(N)$ time sequential algorithm. Kim [15] proposed an $\mathrm{O}(\log N)$ time algorithm for this problem on the EREW PRAM model using $N/\log N$ processors. On the CRCW PRAM model, Berkman et al. [2] also proposed two algorithms; one ran in $\mathrm{O}(\log\log N)$ time using $N/\log\log N$ processors, the other ran in O(1) time using $\mathrm{O}(N^2)$ processors, respectively. For the broadcast-based network, Olariu et al. [25] proposed an O(1) time algorithm for this problem on a 2-D $N \times N$ reconfigurable mesh. Recently, Bhagavathi et al. [3] derived on $\mathrm{O}(\log N)$ time algorithm for this problem on a 2-D MCCMB using $N \times N$ processors.

As the same as solving ANN problem presented in previous subsection. For each element a_i, $0 \leqslant i < N$, we can use N processors to find the lm and rm of it, respectively. Then the ANSV problem can be also solved in O(1) time on a 2-D HBBN using $N \times N$ processors. Initially, assume that these N elements are stored in processor $P_{0,j}$, $0 \leqslant j < N$. Finally, the lm and rm of each a_j are stored in the local variables $\mathrm{lm}(0,j)$ and $\mathrm{rm}(0,j)$ of processor $P_{0,j}$, respectively. Our algorithm consists of the following six major steps. Assume $a_0 = 1$, $a_1 = 3$, $a_2 = 2$ and $a_3 = 0$. A snapshot for solving the ANSV problem is shown in Fig. 5. Step 1, processor $P_{0,j}$, $0 \leqslant j < N$, copies a_j to $P_{i,j}$, $0 \leqslant i < N$, through the i-dimension global bus; then processor $P_{i,i}, 0 \leqslant i < N$, copies a_j to $P_{i,j}, 0 \leqslant j < N$, through the j-dimensional global bus. Thus, each processor $P_{i,j}$ holds two values a_i and a_j, respectively. Step 2, processor $P_{i,j}$, $0 \leqslant j < i$ and $0 \leqslant i < N$, sets $f(i,j) = j$ and its state in active, if $a_j < a_i$; sets $f(i,j) = -\infty$ and its state in inactive, otherwise. Step 3, for each row i, $0 \leqslant i < N$, processor $P_{i,j}$, $0 \leqslant j < i$, applies Corollary 1 on $f(i,j)$ to find the maximal index for all $a_j < a_i$; and let $\mathrm{lm}(i,0)$ store the final result if it exists; let $\mathrm{lm}(i,0) = \mathrm{nil}$, otherwise. Step 4, processor $P_{i,j}$, $i < j < N$ and $0 \leqslant i < N$, sets $f(i,j) = j$ and its state in active, if $a_j < a_i$; sets $f(i,j) = \infty$ and its state in inactive, otherwise. Step 5, for each row i, $0 \leqslant i < N$, processor $P_{i,j}$, $i < j < N$, applies Corollary 1 on $f(i,j)$ to find the minimal index for all $a_j < a_i$, and let $\mathrm{rm}(i,0)$ store the final result if it exists; let $\mathrm{rm}(i,0) = \mathrm{nil}$, otherwise. Step 6, processor $P_{i,0}$, $0 \leqslant i < N$, copies $\mathrm{lm}(i,0)$ and $\mathrm{rm}(i,0)$ to $\mathrm{lm}(i,i)$ and $\mathrm{rm}(i,i)$ through the j-dimension global bus, respectively; and then processor $P_{j,j}$, $0 \leqslant j < N$, copies $\mathrm{lm}(j,j)$ and $\mathrm{rm}(j,j)$ to $\mathrm{lm}(0,j)$ and $\mathrm{rm}(0,j)$ through the i-dimension global bus, respectively. The time complexity is analyzed as follows.



i \ j	0	1	2	3
1	1	3	2	0
1				
2				
3				

(a)

i \ j	0	1	2	3
1	1 1	3 1	2 1	0 1
1	1 3	3 3	2 3	0 3
2	1 2	3 2	2 2	0 2
3	1 0	3 0	2 0	0 0

(b)

	0	1	2	3
1				
1	0 *AS*			
2	0 *AS*	$-\infty$ *IS*		
3	$-\infty$ *IS*	$-\infty$ *IS*	$-\infty$ *IS*	

(c)

	0	1	2	3
1	*nil*			
1	0			
2	0			
3	*nil*			

(d)

i \ j	0	1	2	3
1		∞ *IS*	∞ *IS*	3 *AS*
1			2 *AS*	3 *AS*
2				3 *AS*
3				

(e)

i \ j	0	1	2	3
1	3			
1	2			
2	3			
3	*nil*			

(f)

	0	1	2	3
1	*nil* 3	0 2	0 3	*nil* *nil*
1				
2				
3				

(g)

AS: Active State

IS: Inactive State

Fig. 5. An example for solving the ANSV problem. (a) Initalization; (b) After Step 1; (c) The $f(i, j)$ and processor's state, after Step 2; (d) The lm(i, 0), after Step 3.

Table 1
The results comparison [a]

Problem	Model	Processors	Time	Bus width	Citation
The leftmost one problem	MCCMB	$N \times N$	$O(N^{1/6})$	$O(\log N)$-bit	[31]
		$N \times N$	$O(N^{1/6})$	$O(\log N)$-bit	[11]
	HBBN	$N \times N$	$O(\log N/\log\log N)$	$O(\log N)$-bit	This paper
			$O(1)$	$N^{1/c}$-bit	
The prefix maxima (minima) problem	MCCMB	$N^{1/2} \times N^{1/2}$	$O(N^{1/6})$	$O(\log N)$-bit	[17]
		$N^{5/8} \times N^{3/8}$	$O(N^{1/8})$	$O(\log N)$-bit	[6]
		$N \times N$	$O(\log N)$	$O(\log N)$-bit	[17]
	GMCCMB	$N^{3/5} \times N^{2/5}$	$O(N^{1/10})$	$O(\log N)$-bit	[7]
	MCCHB	N	$O(\log N)$	$O(\log N)$-bit	[12]
	RN	$N \times M$	$O(\log N/\log M)$	$O(\log N)$-bit	[26]
	HBBN	$N \times N$	$O(\log N/\log\log N)$	$O(\log N)$-bit	This paper
			$O(1)$	$N^{1/c}$-bit	
The *m*-contour problem	RN	$N \times M$	$O(\log N/\log M)$	$O(\log N)$-bit	[26]
	HBBN	$N \times N$	$O(\log N/\log\log N)$	$O(\log N)$-bit	This paper
			$O(1)$	$N^{1/c}$-bit	
The ANN problem	CREW	N	$O(\log N)$	$O(\log N)$-bit	[8]
	CRCW	$N/\log\log N$	$O(\log N \log N)$	$O(\log N)$-bit	[29]
		$N^{2+1/c}$	$O(1)$	$O(\log N)$-bit	[29]
	MCCMB	$N \times N$	$O(\log N)$	$O(\log N)$-bit	[26]
	HBBN	$N \times N$	$O(\log N/\log\log N)$	$O(\log N)$-bit	This paper
			$O(1)$	$N^{1/c}$-bit	
The ANSV problem	EREW	$N/\log N$	$O(\log N)$	$O(\log N)$-bit	[15]
	CRCW	$N/\log\log N$	$O(\log\log N)$	$O(\log N)$-bit	[2]
		N^2	$O(1)$	$O(\log N)$-bit	[2]
	MCCMB	$N \times N$	$O(\log N)$	$O(\log N)$-bit	[3]
	HBBN	$N \times N$	$O(\log N/\log\log N)$	$O(\log N)$-bit	This paper
			$O(1)$	$N^{1/c}$-bit	

[a] Note: The computation power of HBBN is the weakest computation model among all models listed in Table 1.

Step 1, 2, 4 and 6 take O(1) time. Steps 3 and 5 each takes O(1) time by Corollary 1. Therefore, the total time complexity is O(1). The following theorem is derived.

Theorem 5. *Given a data sequence $a_0, a_1, \ldots, a_{N-1}$ of N elements from a totally ordered domain, the ANSV problem can be solved in* O(1) *time on a* 2-D HBBN *using $N \times N$ processors, where the bus width of the global bus is $N^{1/c}$-bit ($N^{1/c} > \log N$) for a constant c and $c \geqslant 1$.*

5. Concluding remarks

As we can see, the broadcast-based network is quite suitable for reducing the long distance communications in the existing interconnected parallel processing systems. The HBBN is one of the broadcast-based networks. It is quite practically implementable as all processors are only linked by broadcast buses. To demonstrate the computation power of the HBBN, several interesting fundamental data movement problems have been derived in this paper. These include the leftmost one problem, the prefix maxima/minima problem, the *m*-contour problem, the all nearest neighbor problem and the all nearest smaller values problem which are the most popular and important problems in image processing, digitized geometry and computer graphics. Most of the results derived in this paper are far better than those derived in the other broadcast-based networks. The detailed comparisons are listed in Table 1. Furthermore, all proposed algorithms designed on the HBBN under the O($\log N$)-bit natural bus width can be also executed with the same efficiency on the other broadcast-based networks undoubtedly.

Acknowledgements

The work was supported by National Science Council under the contract no. NSC-85-2221-E011-004.

References

[1] A. Aggarwal, Optimal bounds for finding maximum on array of processors with *k* global buses, IEEE Trans. Comput. 35 (1986) 62–64.

[2] O. Berkman, B. Schieber, U. Vishkin, Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values, J. Algorithms 14 (1993) 344–370.

[3] D. Bhagavathi, V. Bokka, H. Gurla, S. Olariu, J.L. Schwing, I. Stojmenovic, J. Zhang, Time-optimal visibility-related algorithms on meshes with multiple broadcasting, Int. Parallel Processing Symp. (1994) 110–114.

[4] S.H. Bokhari, Finding maximum on an array processor with a global bus, IEEE Trans. Comput. 33 (1984) 133–139.

[5] P. Chaudhuri, Parallel Algorithms: Design and Analysis, Prentice-Hall, Reading, 1992.

[6] Y.C. Chen, W.T. Chen, G.H. Chen, J.P. Sheu, Designing efficient parallel algorithms on mesh-connected computers with multiple broadcasting, IEEE Trans. Parallel and Distributed Systems 1 (1990) 241–246.
[7] K.L. Chung, Prefix computations on a generalized mesh-connected computers with multiple buses, IEEE Trans. Parallel and Distributed Systems 6 (1995) 196–200.
[8] R. Cole, M.T. Goodrich, Optimal parallel algorithms for point-set and polygon problems, Algorithmica 7 (1992) 3–23.
[9] R. Dechter, L. Kleinrock, Broadcast communications and distributed algorithms, IEEE Trans. Comput. 35 (1986) 210–219.
[10] T.Y. Feng, A survey of interconnection networks, IEEE Computer (1981) 12–27.
[11] H. Gurla, Leftmost one computation on meshes with row broadcasting, Inform. Processing Lett. 47 (1993) 261–266.
[12] S.J. Horng, Generalized mesh-connected computers with hyperbus broadcasting for a computer network, IEICE Trans. Inform. Syst. E79-D(8) (1996) 1107–1115.
[13] K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, McGraw-Hill, Reading, 1993.
[14] T.W. Kao, S.J. Horng, Efficient algorithms for computing two nearest neighbor problems on a RAP, Pattern Recognition 27 (1994) 1707–1716.
[15] S.K. Kim, Optimal parallel algorithms on sorted intervals, The 27th Annu. Allerton Conf. Commun., Control, Comput. (1989) 766–776.
[16] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, Benjamin/Cummings, Reading, 1994.
[17] V.K.P. Kumar, C.S. Raghavendra, Array processor with multiple broadcasting, J. Parallel and Distributed Comput. 2 (1987) 173–190.
[18] D.T. Lee, F.P. Preparata, The all nearest neighbor problem for convex polygons, Inform. Processing Lett. 7 (1978) 189–192.
[19] S.P. Levitan, C.C. Foster, Finding an extremum in a network, Int. Symp. Comput. Architecture (1982) 321–325.
[20] H. Li, M. Naresca, Polymorphic-torus architecture for computer vision, IEEE Trans, Pattern Anal. Machine Intell. 11 (1989) 233–243.
[21] J.M. Marberg, E. Gafni, Sorting and selection in multi-channel broadcast networks, Int. Conf. Parallel Processing (1985) 846–850.
[22] M. Maresca, H. Li, Connection autonomy and SIMD computers: A VLSI implementation, J. Parallel and Distributed comput. 7 (1989) 302–320.
[23] R. Miller, V.K.P. Kumar, D. Reisis, Q.F. Stout, Parallel computations on reconfigurable meshes, IEEE Trans. Comput. 42 (1993) 678–692.
[24] S. Olariu, J.L. Schwing, J. Zhang, Data movement techniques on reconfigurable meshes, with applications, Int. J. High Speed Comput. 6 (2) (1994) 311–323.
[25] S. Olariu, J.L. Schwing, J. Zhang, A constant-time channel-assignment algorithm on reconfigurable meshes, BIT 32 (1992) 586–597.
[26] S. Olariu and I. Stojmenovic, Time-optimal proximity algorithms on meshes with multiple broadcasting, Int. Parallel Processing Symp. (1994) 94–101.
[27] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer-Verlog, Reading, 1988.
[28] D.A. Pucknell, K. Eshranghian, Basic VLSI Design, Section 5.3.3, Prentice-Hall, Reading, 1994, pp. 134–138.
[29] B. Schieber, U. Vishkin, Finding all nearest neighbors for convex polygons in parallel: A new lower bound technique and matching algorithm, Discrete Appl. Math. 39 (1990) 97–111.
[30] D.B. Shu, G. Nash, C. Weems, Image understanding architecture and applications, in: J.L.C. Sanz (Ed.), Advanced in Machine Vision, Springer, New York, 1989, pp. 297–355.
[31] Q.F. Stout, Meshes with multiple buses, The 27th IEEE Symp. Found. Comput. Sci. (1986) 264–273.

[32] B.F. Wang, G.H. Chen, Constant time algorithms for transitive closure and some related graph problems on processor arrays with a reconfigurable bus system, IEEE Trans. on Parallel and Distributed Systems 1 (1990) 500–507.
[33] C.B. Yang, R.C.T. Lee, W.T. Chen, Parallel graph algorithms based upon broadcast communications, IEEE Trans. Comput. 39 (1990) 1468–1472.