Information Processing Letters 45 (1993) 69–73
Elsevier

26 February 1993

Permutations on the Block PRAM

Andrew Chin *

Department of Mathematics, Texas A&M University, College Station, TX 77843, USA

Communicated by M.J. Atallah
Received 12 June 1991
Revised 7 December 1992

Abstract

Chin, A., Permutations on the Block PRAM, Information Processing Letters 45 (1993) 69–73.

In present-day parallel computers, the complexity of permuting N data items in shared memory varies, depending on whether large blocks can be used for communication. The Block PRAM model of Aggarwal, Chandra and Snir is unique among shared-memory models of parallel computation in modeling this phenomenon. We characterize the Block PRAM complexity of some useful classes of permutations, improving known results.

Keywords: Computational complexity; Block PRAM; communication latency; parallel computational complexity; permutation routing

1. Introduction

Distinct processors working together on the same problem need to communicate from time to time. In present-day parallel computers, this communication must take place through a physical network and is subject to considerable latency. Because of this latency, parallel computations are most efficient when large blocks are used for communication.

The Block PRAM [2] was recently introduced by Aggarwal, Chandra and Snir as a model of parallel computation accounting for the effect of communication latency on computational complexity. The model assumes that interprocessor communication takes place through shared memory locations, thereby abstracting the issue of communication latency away from the topology of the interprocessor network.

Block PRAM description. A Block PRAM is a collection of p processors, each with a local memory, together with a shared memory. All processors execute the same program, although a processor may wait instead of executing a given instruction. Each arithmetic operation and access to a local memory location can be performed in unit time. Accesses to shared memory are subject to a delay l due to communication latency, where l is a multiple of instruction cycles. A processor may access a block of b consecutive locations in the global memory in time $l + b$. No read or write conflicts are allowed: concurrent requests for overlapping blocks are serviced sequentially in some arbitrary order.

The natural data structure in the Block PRAM is the user-defined array, consisting of a block of shared memory locations with consecutive logical

Correspondence to: Professor A. Chin, Department of Mathematics, Texas A&M University, College Station, TX 77843, USA.

* Research supported by a National Science Foundation Graduate Fellowship and a Rhodes Scholarship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation or the Rhodes Trust.

addresses. Data in the Block PRAM shared memory is addressed according to its location in the relevant array. To create blocks of data for communication across the shared memory, it is often necessary to permute the data in an array.

In this paper, we characterize the complexity of performing specific frequently used permutations on the Block PRAM. As in [2], it is assumed that the permutation to be performed is known in advance, so that the only operation required is the movement of the array data in the shared and local memories. The resulting algorithms are said to be *conservative*. A computation is conservative if the only operation allowed is that of copying elements in memory.

For convenience throughout this paper, we assume p and l are integral powers of 4, affecting our bounds by at most a constant factor. All logarithms will be base 2.

2. Rational permutations

A permutation Π on $\{0, 1, \ldots, 2^k - 1\} \cong \{0, 1\}^k$ is *rational* [1] if it can be expressed as a permutation π in the bit positions; i.e., $\Pi((x_1, \ldots, x_k)) = (x_{\pi(1)}, \ldots, x_{\pi(k)})$. By convention, we write the most significant bit first. If a rational permutation is denoted by a capital Greek letter, we denote its associated bit permutation by the corresponding lower-case Greek letter.

Rational permutations are the permutations most often studied in the literature as network routing problems. Matrix transpositions and perfect shuffles are examples of rational permutations. Standard deterministic permutation algorithms on specific networks [3,7,9–12], and the optimal Block PRAM permutation algorithm [2] amount to factoring an arbitrary permutation into a product of rational permutations.

In this section, we show that the complexity of performing a given rational permutation on the Block PRAM can be characterized in terms of its *crossing number*, as defined below.

Definition. Let Π be a rational permutation on $\{0, 1, \ldots, 2^k - 1\}$ with bit permutation π. Denote $K = \{1, \ldots, k\}$, $C = \{1, \ldots, \log p\}$, and $F = \{k - \log l + 1, \ldots, k\}$. Let $C(\Pi) = \{i \in C: \pi(i) \notin C\}$ and $F(\Pi) = \{i \in F: \pi(i) \notin F\}$. Define $c(\Pi)$, the *coarse crossing number* of Π, by $c(\Pi) = |C(\Pi)|$ and $f(\Pi)$, the *fine crossing number* of Π, by $f(\Pi) = |F(\Pi)|$. Define $\mu(\Pi)$, the *crossing number* of Π, by $\mu(\Pi) = \min(c(\Pi), f(\Pi))$.

Theorem 2.1. *Let Π be a rational permutation. The Block PRAM complexity of performing Π conservatively on $n = 2^k$ consecutive locations in shared memory is*

$$\Theta(n/p + n\mu(\Pi)/(p \log(2n/lp))) \quad \text{if } lp \leqslant n,$$
$$\Theta(l + l\mu(\Pi)/\log(2lp/n)) \quad \text{if } lp > n.$$

This theorem improves results in [2] by giving complexity bounds for each rational permutation rather than worst-case bounds for the class of rational permutations. We prove first the upper bound and then the lower bound.

Lemma 2.2. *Let Π be a rational permutation. A Block PRAM can perform Π conservatively on $n = 2^k$ consecutive locations in shared memory in time*

$$O(n/p + n\mu(\Pi)/(p \log(2n/lp))) \quad \text{if } lp \leqslant n,$$
$$O(l + l\mu(\Pi)/\log(2lp/n)) \quad \text{if } lp > n.$$

Proof. *Case* 1: $lp \leqslant n$. We may assume $lp \leqslant n/2$, affecting our bounds by at most a constant factor. We perform a sequence of basic rational permutations of one of the following forms:

- Rational permutations Σ where $\sigma(i) = i$ for $i \in C$. Each processor reads and permutes a block of size n/p and writes it back.
- Rational permutations Σ where $\sigma(i) = i$ for $i \in F$. Each processor reads n/lp blocks of size l and writes them into their new locations.

Each basic permutation can be performed in time $O(n/p)$.

Assume $c(\Pi) \leqslant f(\Pi)$; the case $f(\Pi) < c(\Pi)$ is analogous. Let $S = K \setminus (C \cup F) \neq \emptyset$. In one basic permutation, any $|S|$ (or fewer) bit positions can be moved from $C(\Pi)$ into S; in a second basic permutation, they can be moved to their images

under π. When all of $C(\Pi)$ has been moved in this way, two more basic permutations suffice to map the remaining positions in C and $K \setminus C$ to their images under π. The number of basic permutations required is

$$2\lceil c(\Pi)/|S|\rceil + 2 = O(c(\Pi)/\log(2n/(lp))).$$

Case 2: $lp > n$. We may assume $lp \geqslant 2n$, affecting our bounds by at most a constant factor. We use the same basic permutations as above. Each basic permutation can be performed in time $O(l)$.

Assume $c(\Pi) \leqslant f(\Pi)$; the case $f(\Pi) < c(\Pi)$ is analogous. Let $T = C \cap F \neq \emptyset$. In one basic permutation, any $|T|$ (or fewer) bit positions can be moved from $\pi^{-1}(C(\Pi))$ into T; in a second basic permutation, they can be moved to their images under π. When all of $\pi^{-1}(C(\Pi))$ has been moved in this way, two more basic permutations suffice to map the remaining bit positions in C and $K \setminus C$ to their images under π. The number of basic permutations required is

$$\begin{aligned} &2\lceil |\pi^{-1}(C(\Pi))|/|T|\rceil \\ &\quad = 2\lceil c(\Pi)/|T|\rceil + 2 \\ &\quad = O(c(\Pi)/\log(2lp/n)). \quad \square \end{aligned}$$

The lower bound uses a potential function argument first used in [2] to prove the lower bound for transposing a square matrix on the Block PRAM. The argument can be stated in the following definition and lemma.

Definition. Let $A = \{0, 1, \ldots, n-1\}$, let $m \mid n$ and let A be divided into n/m *segments* $A_0, \ldots, A_{n/m-1}$ each of length m: $A_i = \{im, \ldots, (i+1)m - 1\}$. Let Π be any (not necessarily rational) permutation on A. For $0 \leqslant r,s \leqslant n/m - 1$, let $x_{r,s}(\Pi, m) = |\Pi(A_r) \cap A_s|$. ($x_{r,s}(\Pi, m)$ denotes the number of elements that are mapped from A_r to A_s.) Define $\Phi(\Pi, m)$, the *m-wise potential* of Π, by

$$\Phi(\Pi, m) = \sum_{s=0}^{n/m-1} \sum_{r=0}^{n/m-1} x^+_{r,s}(\Pi, m) \times \log x^+_{r,s}(\Pi, m)$$

(the notation $x^+_{r,s}$ indicates that the sum is taken over positive $x_{r,s}$ only).

Lemma 2.3. *Let Π be any (not necessarily rational) permutation on $\{0, 1, \ldots, n-1\}$. Any conservative Block PRAM algorithm for performing Π on n consecutive locations in shared memory requires time*

$$\Omega(n/p + (n \log l - \Phi(\Pi, l))/(lp \log(2n/(lp))))$$

for $lp \leqslant n$ and $l \leqslant p$,

$$\Omega(n/p + (n \log p - \Phi(\Pi, p))/(lp \log(2n/(lp))))$$

for $lp \leqslant n$ and $p < l$,

$$\Omega(l + l(n \log(n/p) - \Phi(\Pi, n/p))/(n \log(2lp/n)))$$

for $lp > n$ and $l \leqslant p$,

$$\Omega(l + l(n \log(n/l) - \Phi(\Pi, n/l))/(n \log(2lp/n)))$$

for $lp > n$ and $p < l$.

Proof (sketch). Our notation $\Phi(\Pi, m)$ is the same as the initial value of the potential function in the proof of [2, Theorem 3.3]; the final value of the potential function must be $n \log m$. Without loss of generality, any conservative Block PRAM algorithm can be divided into distinct, alternating read rounds and write rounds each taking time $\Theta(l)$ to access the shared memory. The increase in the potential after r rounds is shown in [2] to be at most $rpl \log(2n/(lp))$ for $lp \leqslant n$, $m = \min(l, p)$, and at most $rn(\log(2lp/n))$ for $lp > n$, $m = \min(n/l, n/p)$. $\square$

Lemma 2.4. *Let Π be a rational permutation on $\{0, 1, \ldots, n-1\}$ with $n = 2^k$. Let $m = 2^j$ with $j < k$. Let $G(m) = \{k - j + 1, \ldots, k\}$, let $G(\Pi, m) = \{i \in G: \pi(i) \notin G(m)\}$ and let $g(\Pi, m) = |G(\Pi, m)|$. Then $\Phi(\Pi, m) = n \log m - ng(\Pi, m)$.*

Proof. Let $r,s \in \{0, \ldots, 2^{k-j} - 1\} \cong \{0, 1\}^{k-j}$. Then by a simple counting argument, $x_{r,s}(\Pi, m) = |\Pi(A_r) \cap A_s|$

$= 0$ if there is an i with $1 \leqslant i, \pi(i) \leqslant k - j$ and the ith bit of r is not $s(\pi(i))$;

$= 2^{j - g(\Pi, m)}$ otherwise.

Moreover, for a given s, there are exactly $2^{g(\Pi,m)}$ choices of r for which $x_{r,s}(\Pi, m)$ is nonzero. Hence

$$\begin{aligned}\Phi(\Pi, m) &= \sum_{s=0}^{n/m-1} \sum_{r=0}^{n/m-1} x^{+}_{r,s}(\Pi, m) \log x^{+}_{r,s}(\Pi, m) \\ &= \sum_{s=0}^{n/m-1} 2^{g(\Pi,m)}\left(2^{j-g(\Pi,m)} \log 2^{j-g(\Pi,m)}\right) \\ &= (n/m)2^{j}(j - g(\Pi, m)) \\ &= n \log m - ng(\Pi, m). \quad \square\end{aligned}$$

Corollary 2.5. *Let Π be any rational permutation on $\{0, 1, \ldots, n-1\}$. Any conservative Block PRAM algorithm for performing Π on $n = 2^k$ consecutive locations in shared memory requires time*

$$\Omega(n/p + n\mu(\Pi)/(p \log(2n/lp))) \quad \text{if } lp \leqslant n,$$
$$\Omega(l + l\mu(\Pi)/\log(2lp/n)) \quad \text{if } pl > n.$$

This completes the proof of Theorem 2.1.

Corollary 2.6. *The Block PRAM complexity of transposing an $a \times b$ matrix conservatively on $n = 2^k$ consecutive locations in shared memory is*

$$\Theta\left(n/p + n \log \min(p, l, a, b) / (p \log(2n/(lp)))\right) \quad \text{if } lp \leqslant n,$$
$$\Theta(l + l \log \min(p, l, a, b)/\log(2lp/n)) \quad \text{if } lp > n.$$

The crossing numbers of some frequently used examples of rational permutations are evaluated in Table 1. For convenience, we assume k is even.

3. Conclusions

A unique feature of the Block PRAM as a shared-memory model of parallel computation is that it provides a complexity theory of permutations. In this paper, we have characterized the Block PRAM complexity of important classes of permutations.

Our results can be extended. Consider the *linear permutations* Π_M on $\{0, 1\}^k$ which defined by $\Pi_M(x) = Mx$, where M is a nonsingular $k \times k$ 0–1 valued matrix and arithmetic is modulo 2. Up to a constant factor, linear permutations are no harder to perform than the most difficult rational permutations (e.g., bit reversal), see [4,5]. Rational permutations are a special case of linear permutations where M is a permutation matrix, so that this result is tight. Linear permutations have applications to hashing [4,8] and skewing [5,6] techniques, which provide for parallel memory access.

The potential function argument used in this paper (Lemma 2.3) is essentially information-theoretic, and there is much room for improved lower bounds for specific permutations. We believe that this complexity theory of permutations has intrinsic combinatorial interest as well as application to the development of high performance, massively parallel computers.

Table 1

Permutation Π	π	$\mu(\Pi)$
Identity	$[1, \ldots, k]$	0
Perfect shuffle	$[k, 1, \ldots, k-1]$	1
jth power of shuffle	$[k-j+1, \ldots, k-j]$	$\min(\log p, \log l, \lvert j\rvert, k - \lvert j\rvert)$
$r \times s$ matrix transpose	$[k \cdot \log r, \ldots, k - \log r - 1]$	$\log \min(p, l, r, s)$
$n^{1/2} \times n^{1/2}$ transpose	$[k/2, \ldots, k/2-1]$	$\log \min(p, l)$
Bit reversal	$[k, \ldots, 1]$	$\log \min(p, l)$
Bit shuffle	$[1, 3, \ldots, k-1, 2, 4, \ldots, k]$	$\log \min(p, l)$
Shuffled row-major	$[1, k/2+1, 2, \ldots, k]$	$\log \min(p, l)$
j-way shuffled row-major	$[1, k/j+1, 2k/j+1, \ldots, k]$	$\log \min(p, l)$

References

[1] A. Aggarwal, A.K. Chandra and M. Snir, Hierarchical memory with block transfer, in: *Proc. 28th Ann. IEEE Symp. on Foundations of Computer Science* (1987) 204–216.

[2] A. Aggarwal, A.K. Chandra and M. Snir, On communication latency in PRAM computations, in: *Proc. First Ann. ACM Symp. on Parallel Algorithms and Architectures* (1989) 11–21.

[3] K. Batcher, Sorting networks and their applications, in: *Proc. AFIPS Spring Joint Computing Conf.* **32** (1968) 307–314.

[4] A. Chin, Locality-preserving hash functions for general purpose parallel computation, *Algorithmica*, to appear.

[5] A. Chin, Practical issues in parallel complexity, D. Phil. thesis, University of Oxford, August 1991, in preparation.

[6] K. Kim and V.K. Prasanna Kumar, Perfect Latin squares and parallel array access, in: *Proc. 16th Ann. IEEE-ACM Internat. Symp. on Computer Architecture* (1989) 372–379.

[7] J. Lenfant, Parallel permutations of data: a Benes network control algorithm for frequently used permutations, *IEEE Trans. Comput.* **27** (1978) 637–647.

[8] K. Mehlhorn and U. Vishkin, Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories, *Acta Inform.* **21** (1984) 339–374.

[9] D. Nassimi and S. Sahni, An optimal routing algorithm for mesh-connected parallel computers, *J. ACM* **27** (1980) 6–29.

[10] M.C. Pease III, The indirect binary n-cube microprocessor array, *IEEE Trans. Comput.* **26** (1977) 458–473.

[11] C.S. Raghavendra and V.K. Prasanna Kumar, Permutations on Illiac IV-type networks, *IEEE Trans. Comput.* **35** (1986) 662–669.

[12] H.S. Stone, Parallel processing with perfect shuffle, *IEEE Trans. Comput.* **20** (1971) 153–161.