Routing on the PADAM: Degrees of Optimality *

Bogdan S. Chlebus[†] Artur Czumaj[‡] Jop F. Sibeyn[§]

Abstract

Routing problems are studied for the Parallel Alternating-Direction Access Machine. The goal is to investigate what level of optimality can be achieved depending on loads of packets per memory unit. In the case of typical moderate loads, our algorithms are optimal to within a small multiplicative constant; a deterministic and a randomized algorithm are developed, both faster than the best previously known routing algorithm. Moreover, for sufficiently large loads, an algorithm which misses optimality by only an additive lower-order term is designed. We consider also off-line routing problems, and multidimensional extension of the model.

1 Introduction

The Parallel Alternating-Direction Access Machine, *PADAM*, is a multiprocessor computer with memory units, *MUs*, arranged in a grid structure, together with the processing units, *PUs*. The two-dimensional $n \times n$ PADAM has n PUs and n^2 MUs arranged according to the pattern in Figure 1 (notice that there are only n PUs: the PUs marked with the same number are identical).

The PADAM shares features from both ordinary meshes and from PRAMs. The essential differences with meshes are the following:

1. A two-dimensional $n \times n$ mesh consists of n^2 PUs, n times as many as the corresponding PADAM.
2. A PU in a mesh can communicate only with its immediate neighbors, whereas, in principle, a PU in a PADAM can communicate with any other PU in constant time.

*This work was supported by the following contracts: DFG-Graduiertenkolleg "Parallele Rechnernetzwerke in der Produktionstechnik" Me 872/4-1, DFG-SFB 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen", DFG Leibniz Grant Me872/6-1, and EU ESPRIT Long Term Research Project 20244 (ALCOM-IT). This research was partly done while the first and third authors were visiting Heinz Nixdorf Institut, Universität-GH Paderborn.

[†]Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02-097 Warszawa, Poland. Email: chlebus@mimuw.edu.pl.

[‡]Heinz Nixdorf Institut und Fachbereich Mathematik/Informatik, Universität-GH Paderborn, 33095 Paderborn, Germany. Email: artur@uni-paderborn.de.

[§]Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. WWW: http://www.mpi-sb.mpg.de/~jopsi/. E-mail: jopsi@mpi-sb.mpg.de.

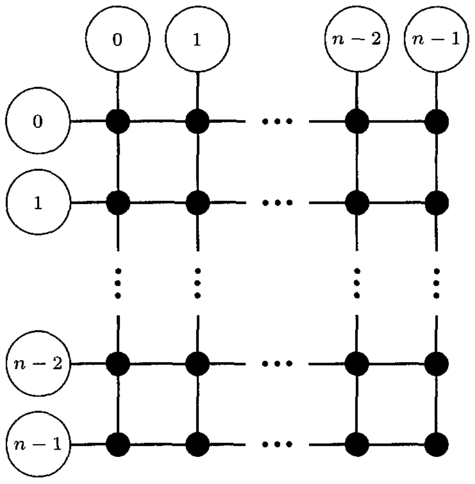

Figure 1: A PADAM with n PUs and an $n \times n$ network of MUs. The large circles indicate the PUs, the smaller circles the MUs. A PU has direct access to all MUs in its row and column.

Two existing machines, ADENA and USC/OMP, were direct inspirations for the PADAM design. For the technical details of ADENA see [7, 8], and for UCS/OMP see [4, 5]. The ADENA and USC/OMP were built primarily for numerical, scientific and computer-graphics applications. In these problems, arrays of numbers need to be processed, and such arrays map naturally onto arrays of MUs.

The formal PADAM model we consider abstracts from low-level technical details. It was introduced in [2], where the potential of this architecture as a general-purpose computer was explored. General applications require greater flexibility of accessing memory and rearranging information. To facilitate such tasks, in [2] a routing algorithm was designed, mechanisms of random access to MUs were developed, and general PRAM simulations were given.

This research concentrates on the problem of routing. The goal is to investigate the levels of optimality that can be achieved for various loads of the network. By a *load* we mean the number r of packets from every MU that have to be relocated. For the smallest loads, the results are away from off-line optimal by a logarithmic factor, while for $r = \omega(\log n \cdot \log\log n)$, the difference decreases to an additive lower-order term.

2 Preliminaries

In this section we specify some parameters of the model and introduce the routing problem.

Indexing. We restrict our attention to 2-dim PADAM, unless stated otherwise. On an $n \times n$ PADAM, there are n PUs: $PU_0, \ldots, PU_{n-1}$, and n^2 MUs: $MU[x, y]$, for all $0 \leq x, y < n$.

Modes of Operation. The PUs can work in three modes: *internal mode*, *row mode* and *column mode.* At any time, all PUs operate in the same mode. In the internal mode, the PUs perform internal computations; in the row mode each PU can access one MU of its choice in its row; and in the column mode each PU can access a MU in its column. We assume that synchronization is provided by the model: when a processor wants to switch the row/column mode, it enters a state of being ready for this, and when all processors are in this state then the mode is changed.

Cost Model. The internal processor computations are assumed to take negligible time. The cost of a memory operation consists of two parts: the time to access a MU (latency), followed by the time to perform a number of reads and writes in the MU (transfer time). To simplify considerations, we consider one of these costs for a problem at hand. Usually for on-line algorithms we count MU accesses, and for off-line algorithms we count writes.

Routing Problem. We consider the problem of routing *r-relations* (among MUs): each MU is the source and destination of (at most) r packets, and the goal is to route the packets to their destination MUs efficiently (if $r = 1$ then the problem is referred to as *permutation routing*). The given basic formulation of the problem is memory oriented. We also consider a processor-oriented formulation of routing *R-relations* (among PUs), in which every PU is the source and destination of $R = r \cdot n$ packets (the packets of PUs are parked in the hyperplanes assigned to PUs).

3 Deterministic Algorithm for Moderate Loads

In this section the problem is of routing r-relations among MUs, the cost is measured as the number of accesses to different MUs. The exposition concentrates on the case $r = 1$, extension to the general case being straightforward.

A sequence $\mathbf{x} = \langle x_0, \ldots, x_{n-1} \rangle$ of numbers in $[0..n-1]$ is an *n-sequence.* If $x_i = s$ then x_i is referred to as *s-value.* We start with a sequential algorithm to permute n-sequences. The input is an n-sequence $\mathbf{x}$, where n is a power of 2, and the output is a permutation of $\mathbf{x}$ stored in array $X[0..n-1]$.

```
Algorithm DISPERSE

1. Sort x.
2. If $n = 1$ then place $x_0$ in $X[0]$; otherwise:
    (a) apply Step 2 recursively to $\mathbf{x}_l = \langle x_0, x_1, \ldots, x_{n/2-1} \rangle$ in array $X_l$ consisting of $X[0], X[2], X[4], \ldots$ ;
    (b) apply Step 2 recursively to $\mathbf{x}_r = \langle x_{n/2}, x_{n/2+1}, \ldots, x_{n-1} \rangle$ in array $X_r$ consisting of $X[1], X[3], X[5], \ldots$.
```

Lemma 1 *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be two n-sequences such that the number of s-values in* $\mathbf{x}$ *and t-values in* $\mathbf{y}$ *are equal to $\beta > 0$. Apply the algorithm* DISPERSE *on* $\mathbf{x}$ *to obtain* $\mathbf{x}'$ *and on* $\mathbf{y}$ *to obtain* $\mathbf{y}'$*. Let $x'_{i_1}, \ldots, x'_{i_\beta}$ be the consecutive s-values* *in* $\mathbf{x}'$. *Then, for each segment* I *among* $[0..i_1]$, $[i_1+1..i_2]$, ..., $[i_{\beta-1}+1..i_\beta]$ *and* $[i_\beta+1..n-1]$ *the number of t-values* $\mathbf{y}'_i$ *such that* $i \in I$ *is at most four.*

Algorithm A1**:**

1. In row mode: Each PU_i counts the number of packets in its row, which are to be delivered to column j, for each $j \in [0..n-1]$. This number is stored in $M[i,j]$. The column address is the *value* of a packet.
2. In column mode: Each PU_i reads in $M[k,i]$, for each k, the number t of packets in row k going to column i, and creates t *dummy packets* of value k. These packets are stored in the column one packet per memory unit.
3. In row mode: Each PU_i disperses the packets in row i with respect to values.
4. In column mode: Each PU_i disperses the dummy packets in column i with respect to their values.
5. Each PU_i executes the following, for $m := 1$ to n:
   (a) In row mode: It takes the m-th packet and delivers it to its destination column.
   (b) In column mode: It reads the value of the m-th dummy packet, let it be k. Then it goes to row k and delivers (at most four) packets waiting there to their destination rows.
6. In column mode: Each PU_i scans the column list and delivers all the waiting packets to their destination columns.

Lemma 2 *Algorithm* A1 *is correct and delivers all the packets.*

To have small constants, the implementation of algorithm A1 can be simplified. The rows are first sorted by a bucket sort. In the course of this the numbers of packets in row i going to column j are calculated and stored in $M[i,j]$. The permutation of algorithm DISPERSE does not need to be performed, because it is fixed: instead we calculate the original addresses of packets that would be moved by the algorithm DISPERSE to specific locations.

Lemma 3 *After sorting, algorithm* DISPERSE *moves a packet from position* i *to* j *in the sequence iff the binary representation of* j *is the reversed binary representation of number* i.

Hence, if a processor needs to access a packet that would be placed at address l by DISPERSE then it takes the packet from the original location with address obtained by reversing the bits of number l.

The modified version of algorithm consists of three phases: (1) sorting the packets in rows; (2) creating and distributing dummy packets in columns; (3) delivering packets.

Theorem 1 *Algorithm* A1 *can be implemented to run in time* $18n$, *measured as the number of accesses to MUs.*

4 Randomized Algorithm for Moderate Loads

We develop a randomized algorithm for permutation routing among memory units, which has a better constant in the expected-performance bound than A1 in its worst-case performance bound. The cost is again the number of accesses to different MUs.

Algorithm A2**:**

1. In column mode: Each PU_i permutes randomly the packets in column i.
2. In row mode: Each PU_i builds a *row list* R_i in row i: this list connects columns to which there are still packets to be delivered from row i. For each column k in R_i, all the MUs storing packets addressed to k are connected in a list, called *row-column group* $G(i, k)$, the header of this list is placed in $M[i, k]$.
3. In column mode: Each PU_k builds a *column list* C_k: this list connects such MUs $M[i, k]$ that in row i there are still some packets that need to be moved to column k.
4. Repeat for each PU, until routing completed:
   (a) In row mode: PU_i scans the row list R_i. For each considered column k, a packet x from the row-column group $G(i, k)$ is retrieved, its MU is deleted from $G(i, k)$, and packet x is stored temporarily in $M[i, k]$. (Such a packet x is a *waiting packet.*) If $G(i, k)$ is empty then column k is deleted from list R_i.
   (b) In column mode: PU_k scans the column list C_k. For each processed $M[i, k]$ in C_k, PU_k delivers the waiting packet stored there, if any, to its final destination MU. If $G(i, k)$ is empty then row i is deleted from list C_k.

Theorem 2 *For each constant $\alpha > 0$ and sufficiently large n, Algorithm* A2 *operates in time $(12 - e^{-1} + o(1)) \cdot n$ with the probability at least $1 - n^{-\alpha}$.*

Faster randomized implementation. Observe that Algorithm A2 performs many memory accesses for list operations, and in particular, it requires an expensive preprocessing in Steps 2 and 3. One can obtain a faster implementation by working with lists maintained implicitly.

Theorem 3 *For each constant $\alpha > 0$ and sufficiently large n, Algorithm* A2 *can be implemented to operate in time $(7 + e^{-1} + o(1)) \cdot n$ with the probability at least $1 - n^{-\alpha}$.*

5 Algorithm for Large Loads

In this section the cost measure is the number of write operations, again we consider routing r-relations among MUs. For sufficiently large r, we want to match the time consumption of the off-line algorithm up to lower-order *additive* terms.

Consider first the problem of off-line routing r-relation on an $n \times n$ PADAM. It can be performed by first solving a coloring problem for a bipartite (multi-) graph. The n nodes on the left correspond to the rows, the n nodes on the right to the columns. There is an edge from a node i to a node j for each packet in row i with destination in column j. All nodes are of degree nr, and hence the graphs can be colored using nr colors. The colors corresponds to the steps in which a packets is going to be routed: Consider a packet p with color c, $0 \leq c < nr$, standing in row i and with destination in column j. In Step c, PU_i writes p to $MU[i,j]$, and subsequently, p is read by PU_j and is written into its destination MU. The bound in the next Lemma 4 is tight.

Lemma 4 *On a two-dimensional PADAM, r-relations can be routed off-line in $2nr$ steps.*

Our approach for on-line routing for larger load is to simulate the off-line algorithm above. We need to solve a coloring problem on a bipartite regular (multi-) graph *on-line.* This problem involves n nodes in both components, and $n^2 \cdot r$ edges between them. On-line algorithms along these lines have been designed before for meshes [10].

A bipartite regular graph with v nodes and m edges can be colored sequentially in $\mathcal{O}(m \cdot \log(m/v))$ steps [3]. Lev, Pippenger, and Valiant [9] designed a PRAM algorithm that uses $m/\log m$ PUs and solves the edge-coloring problem for m/v being a power of two in the optimal time $\mathcal{O}(\log m \cdot \log m/v)$. This algorithm can be implemented on the PADAM without loss provided m is sufficiently large:

Lemma 5 *Edge-coloring a regular bipartite graph with $v = 2n$ nodes, $m \geq n^2 \log\log n$ edges and m/n a power of two, can be performed on an $n \times n$ PADAM in the optimal $\mathcal{O}(m/n \cdot \log(m/n))$ time.*

Substituting $v = n$ and $m = n^2 \cdot r$ shows that it would take $\mathcal{O}(nr \cdot \log(nr))$ time.

In every PU, the packets with the same destination are grouped together, and combined into super-packets of size s. For every destination there may be one super-packet that is only partially filled. Thus, for given r and s, there are at most $\lfloor nr/s \rfloor + n$ super-packets in every PU.

Theorem 4 *For any $r = \omega(\log n \log\log n)$, an $n \times n$ PADAM can route r-relations on-line in $2nr + o(nr)$ steps.*

6 Multidimensional PADAMs

The problem considered is R-relation routing among PUs of a multidimensional PADAM, the cost is the number of writes in MUs.

By a d-dimensional PADAM we mean a d-dimensional grid of MUs, with all PUs assigned to one face of this grid, this face may change in the course of the computation. A PU with index $(i_1, \ldots, i_{d-1})$, denoted $PU[i_1, \ldots, i_{d-1}]$ can write to any $MU[i_1, \ldots, i_{d-1}, x]$, for any $0 \leq x < n$, and only to these MUs. The PUs can read from $d - 1$ sides. The indexing on the sides is such that in an operation in which the reading is along the j-axis, $PU[i_1, \ldots, i_{d-1}]$ can read from $MU[i_1, \ldots, i_{j-1}, x, i_{j+1}, \ldots, i_{d-1}, i_j]$, for all $0 \leq x < n$. In every step every PU can perform an arbitrary internal computation, one write, and one read. The reads have to be performed along the same axis.

Lower-Bound. A *correction* is the action of moving a packet such that one of its coordinates becomes equal to the corresponding coordinate of its destination. If initially no packet shares a coordinate with its destination, then $(d-1) \cdot R \cdot n^{d-1}$ corrections are needed. Dividing by the number of PUs shows that

Lemma 6 *There are inputs for which routing R-relations on a d-dimensional* $n \times \cdots \times n$ *PADAM requires at least* $(d-1) \cdot R$ *steps.*

Only for $d = 2$, this bound is sharp. For $d = 3$, we consider the transposition under which all packets from $PU[i, j]$ have to be routed to $PU[j, i]$. All the packets with the same first coordinate have destination in PUs with the same second coordinate, and vice-versa. Thus, during the first step, only n corrections can be performed. The remaining $2 \cdot R \cdot n^2 - n$ corrections that have to be performed require at least $2 \cdot R$ steps.

Off-line Algorithm. By combining Lemma 4 with the results from [1], we prove:

Theorem 5 *R-relations can be routed off-line on a d-dimensional PADAM in* $(2 \cdot d - 3) \cdot R$ *steps.*

Unfortunately, the gap with the lower-bound of Lemma 6 is large.

On-line Algorithm for Large R. For turning the off-line algorithm into an on-line algorithm, the in total $(d-2) + (2 \cdot d - 3)$ coloring operations must be performed on-line. The costs can be amortized over the $2 \cdot d - 3$ routing rounds. Also the ratio of node, edges and PUs is the same as before. Therefore, the result of Theorem 4 can be carried on:

Corollary 1 *On a d-dimensional* $n \times \cdots \times n$ *PADAM, for* $R = \omega(n \cdot \log n \cdot \log\log n)$, *R-relations can be routed on-line in* $(2 \cdot d - 3) \cdot (R + o(R))$ *steps.*

7 Conclusion

By a detailed analysis of routing on the PADAM, we have quantified the degree of optimality for routing r-relations among memory units and R-relations among processors, that can be realized for various r and R. For r such that $r = o(\log n)$, the routing requires an extra factor $\mathcal{O}(\log n)$, for $r = \mathcal{O}(\log n \cdot \log\log n)$ an extra constant factor, and for larger r only an additive lower-order term.

As routing R-relations among processors constitute the basic communication operation of BSP algorithms, our analysis strengthens the claim that a PADAM is a suitable architecture for general purpose parallel computation. It was already known that PRAM algorithms could be simulated efficiently [2], but for possible practical applications the involved constants are of crucial importance. This last point has been settled now, by showing that even for irregular and non-random patterns full efficiency can be reached for moderate values of R.

For higher dimensional PADAMs, it remains an open problem to either sharpen the lower-bound, or to improve the given algorithms. For $d = 3$, the optimal time appears to be $3 \cdot R$, but for $d \geq 4$, we do not dare to make a guess.

References

[1] F. Annexstein, M. Baumschlag, 'A Unified Approach to Off-line Permutation Routing on Parallel Networks,' *Proc. 2nd ACM Symposium Parallel Algorithms and Architectures*, ACM, 1990, pp. 398–406.

[2] B.S. Chlebus, A. Czumaj, L. Gąsieniec, M. Kowaluk, and W. Plandowski, 'Parallel Alternating-Direction Access Machine,' *Proc. 21st International Symposium on Mathematical Foundations of Computer Science*, Springer-Verlag, 1996, LNCS 1113, pp. 267–278,

[3] R. Cole and R. J. Hopcroft, 'On Edge Coloring Bipartite Graphs,' *SIAM Journal on Computing* 11 (1982) 540–546.

[4] K. Hwang and C-M. Cheng, 'Simulated Performance of a RISC-based Multiprocessor Using Orthogonal-Access Memory,' *Journal of Parallel and Distributed Computing* 13 (1991) 43–57.

[5] K. Hwang, P.-S. Tseng, and D. Kim, 'On Orthogonal Multiprocessor for Parallel Scientific Computations,' *IEEE Transactions on Computers* 38 (1989) 47–61.

[6] J. JáJá, *"An Introduction to Parallel Algorithms"*, Addison Wesley, Reading, MA, 1992.

[7] H. Kadota, K. Kaneko, I. Okabayashi, T. Okamoto, T. Mimura, Y. Nakakura, A. Wakatani, M. Nakajima, J. Nishikawa, K. Zaiki, and T. Nogi, 'Parallel Computer ADENART - its Architecture and Application,' *Proc. of the 5th ACM International Conference on Supercomputing*, 1991, pp. 1–8.

[8] H. Kadota, K. Kaneko, Y. Tanikawa, and T. Nogi, 'VLSI Parallel Computer with Data Transfer Network: ADENA,' *Proc. of the International Conference on Parallel Processing*, Vol. I, 1989, pp. 319–322.

[9] G. F. Lev, N. Pippenger, and L.G. Valiant, 'A Fast Parallel Algorithm for Routing in Permutation Networks,' *IEEE Transactions on Computers* 30 (1981) 93–100.

[10] T. Suel, 'Permutation Routing and Sorting on Meshes with Row and Column Buses,' *Parallel Processing Letters* 5 (1995) 63–80.