Parallel Computing 14 (1990) 229–233
North-Holland

Short Communication

Implementation of matrix multiplication on the T-RACK

T. THEOHARIS
St Catharine's College, University of Cambridge, Cambridge, UK

J.J. MODI
Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, UK

Received June 1989
Revised October 1989

Abstract. It turns out to be advantageous to partition matrices into blocks for multiplication on the T-RACK, which is a MIMD system consisting of 64 floating-point transputers with partially re-configurable linkage.

Keywords. Linear algebra, matrix multiplication, MIMD systems, transputers, performance results.

1. Introduction

The T-RACK [1], which is designed to simulate novel parallel architectures, consists of 64 32-bit floating-point transputers, which may be interconnected by a combination of fixed and program-controlled re-configurable links. This permits a variety of topologies to be configured in hardware, and thus renders the system adaptable to a wide range of applications. A brief discussion of the system is presented in Section 2.

Let A and B be matrices of size $m \times n$ and $n \times p$ respectively. Formation of the matrix product $C = AB$ with elements

$$c_{ij} = \sum a_{ik} b_{kj} \quad 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant p$$

requires mnp products to be calculated. A number of different strategies for forming these products are amenable to implementation on a parallel machine with, say, N processors, and it is useful to distinguish three cases (a) $N > mnp$, (b) $mnp \geqslant N \geqslant \max(mn, np, mp)$, (c) $N < \max(mn, np, mp)$.

In case (a) all multiplications can be performed in parallel in a single multiplication step, but $\lceil \log_2 n \rceil$ addition steps are needed.

For case (b) it is possible to perform mp, mn or np multiplications in parallel, and we usually associate the *outer-product* algorithm with the case $\min(m, n, p) = n$, the *middle-product* algorithm *by rows* with the case $\min(m, n, p) = m$, and the *middle-product* algorithm *by columns* with the case $\min(m, n, p) = p$.

For case (c) the matrices need to be partitioned into smaller blocks, where each block is dealt with in parallel; thus, for example, the matrices A and B and the product may be partitioned as follows:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

where each product on the right-hand side is computed in parallel. This technique is particularly suitable for the T-RACK (see below). Further details on other techniques may be found in various references; see for example [3] or [2].

2. The architecture of the T-RACK

Sixteen cards, each containing four transputers, are placed in a rack. Each transputer holds 1 Mbyte of external dynamic random access memory (100 ns access time), has processor speed of 10 Mips, and possesses four serial communication links operating at 20 Mbits/second. If these links are labelled 0, 1, 2 and 3 and the transputers as $0, \ldots, 63$, then the 1st link of the kth transputer is connected to the 0th link of the $(k+1)$th transputer (the first and last transputers are connected via the interface card). Thus there is a *hamiltonian path* connecting all 64 transputers. In addition the 2nd and 3rd links are connected to two 128×128 crossbar switchboards, which may provide long distance communication between any two transputers. The system is front-ended with a SUN 3/110 colour workstation, and communication between the T-RACK and the SUN is via transputer links to/from a transputer based interface card which is accessed through the VME-bus in the SUN (see Fig. 1). The primary function of the control card is to set up the switch cards to the required configuration; the control card monitors the transputers via the monitoring bus, as indicated in Fig. 1. The system may be run in any programming environment containing an OCCAM 2 compiler on any network of transputers. Furthermore the system is modular, and multiple T-RACK's can easily be configured.

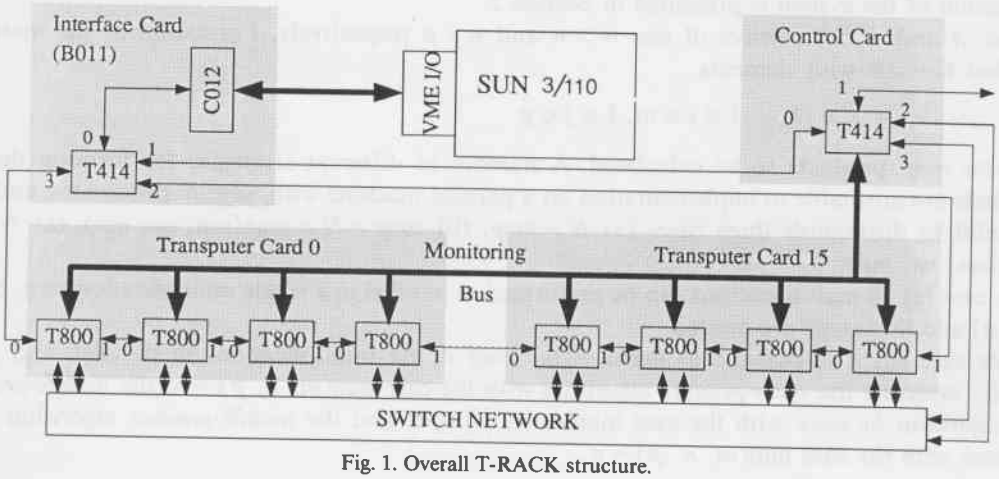

Fig. 1. Overall T-RACK structure.

3. Implementation and results

3.1. On block multiplication

The major cost factors *in time* are computation and communication. Blocks are multiplied in the same manner as matrices, and thus the computational cost of block multiplication is $O(b^3)$, assuming $b \times b$ blocks. The communication cost involved in the transmission of a block from one transputer to another is $O(b^2)$, i.e. proportional to the number of elements in the block.

The advantage of partitioning a matrix into blocks compared with the other methods mentioned in the introduction, is that the ratio of the computation over the communication cost is greater than the corresponding figure for the other methods. It is therefore a more efficient way of distributing the matrices among the transputers. Since the computation cost is an order of magnitude greater than the communication cost, it also follows that there will be a critical block size above which it will be computationally cheaper to perform matrix multiplication by distributing the matrices among the transputers in blocks rather than computing the product on a single transputer. Furthermore, the larger the size of the matrices, the greater the number of block products, and hence the greater the number of transputers that can be usefully employed for the task. In the sequel we shall only consider the implementation of the block partitioning method. A table of experimental timing results for this method based on the T800 transputer is presented in Section 3.3 in support of the above claims.

3.2. Description of the implementation

The $m \times n$ matrix A and the $n \times p$ matrix B are partitioned into $(m+p)n/b^2$ blocks of $b \times b$ elements; the blocks are transmitted to the transputers which we shall refer to as *multipliers* (m, n and p are assumed to be multiples of b). The multipliers are linearly connected using the hardwired transputer links; each of them receives the blocks of A and B from its left neighbour and passes them on to its right neighbour keeping one block of each of A and B. The number of multipliers needed is mnp/b^3, and this number can be varied by changing the block size b. For $i = 0, \ldots, mnp/b^3 - 1$, multiplier i keeps blocks

$$(i \text{ DIV } np/b^2,\ i \text{ REM } n/b) \text{ of } A$$

and

$$(i \text{ REM } n/b,\ (i \text{ DIV } n/b) \text{ REM } p/b) \text{ of } B,$$

where DIV indicates integer division and REM the remainder operator. Blocks are indexed in the same way as matrix elements (row, column). Each of the multipliers then multiplies its local blocks of A and B. The multipliers are then partitioned into mp/b^2 contiguous groups of n/b multipliers each and the product blocks of each group are summed together into the "leftmost" transputer; this can be done in $\lceil \log_2 n/b \rceil$ block addition steps. The resulting blocks, which reside in the "leftmost" transputer of each group, form the product of A and B and are transmitted to the transputer residing on the interface card (Fig. 1).

3.3. Results

In order to demonstrate experimentally our assertion (see Section 3.1) that the larger the matrices to be multiplied the larger the number of processors that can be usefully employed using the block multiplication technique, we timed the multiplication of a number of matrices of varying sizes on 1, 8 and 64 T800 transputers. The results provide an indication of the speedup as well as the absolute speed that can be achieved when this technique is used on

Table 1
Matrix multiplication times on the T-RACK

Matrix size	Block size (b)		Time (ms)		
m ($= n = p$)	8 T800's	64 T800's	1 T800	8 T800's	64 T800's
4	2	1	0.9	0.9	2.9
8	4	2	7	2.8	4.5
16	8	4	54.5	13.4	10.9
32	16	8	430.7	82.9	39.4
64	32	16	3458.5	584.4	179.4
128	64	32	27649.4	4380.4	960.2
256	128	64	220991.4	33802.6	5959.7

transputers. For simplicity we have assumed that $m = n = p$. In Table 1 we list matrix sizes against time required for multiplication on 1, 8 and 64 transputers; the matrices consist of 32-bit real numbers.

In the single-transputer implementation, no communication is involved because the matrices to be multiplied as well as the result are stored within the local memory of the single transputer. In the case of the 8 and 64 transputers, the matrices are partitioned into the appropriate number of blocks by the transputer of the interface card and then communicated to the transputer multipliers; this communication time (i.e. the distribution of the operand blocks to the multipliers as well as the collection of the result blocks by the transputer of the interface card) is taken into account in the above timing figures.

It is seen that for very small matrix sizes (up to 4×4) the cost of the communication is not outweighed by the gains from distributing the computation, and the single-transputer implementation is at least as fast as the 8- or 64- transputer implementations. Somewhere between the matrix sizes of 8×8 and 16×16, the 64-transputer implementation becomes faster than that with 8 transputers. The speedup of the multi-transputer implementations over the single transputer implementation is monotonically increasing with matrix size because the computational complexity of block multiplication increases more rapidly than the number of elements in the block (see Section 3.1). For the 64-transputer implementation the computation time for 256×256 matrices is 6 seconds, and this represents a speedup of 37 over the single-transputer implementation. The substantial reuse of data involved in block multiplication makes this method advantageous over the other partitioning methods mentioned in the introduction, for coarse grain parallel processors with (relatively) slow communication links, such as the transputer.

In the implementation of block communication we had the following options:

(a) whether to use buffering, and

(b) to choose the optimal slice size.

The use of buffering enables more than one transputer link to be simultaneously active (when data is being transferred through a transputer) but requires a considerable setup time. We found that the use of buffering provided a marginal performance gain. The size of the communicated slices could range between one matrix element and a whole block; the smaller the slice size the smaller the synchronisation delays but the larger the number of communication setups. We experimented with slices of sizes ranging from one element (4 bytes) to 256 elements and found that the larger the slice size the better (within the range considered). However the performance gains were inversely proportional to the slice size.

References

[1] P. Capon, J. Gurd and A. Knowles, Parsifal: a parallel simulation facility, in *IEEE Colloquium on the Transputer Application & Case Studies* (1986).
[2] W.M. Gentleman, Some complexity results for matrix multiplication on parallel processors, *J. ACM* **25** (1) (1978) 112–115.
[3] J.J. Modi, *Parallel Algorithms and Matrix Computations* (Oxford University Press, Oxford, 1988).