数理解析研究所講究録
第625巻 1987年 132-136

Exponential Speedup by Vector Operations

Kazuo Iwama
Kyoto Sangyo University

1. Introduction.

To find elementary facilities for parallel computation has been one of the central research goals in the computational complexity theory. A variety of such facilities are known including versatile shared memory in PRAMs [3,4], sophisticated nondeterminism called alternation [1], unbounded fan-in logic gates of combinatorial circuits [2] and so on. All of them were proved to have the same power that gives us the speedup from poly-space to poly-time. Among others, one of the most interesting ones was presented in [5,7] that showed the shift operation for integer (or Bool) vectors gives us the same speedup. In the current paper we present another, similar in a sense, vector operation which allows more speedup, namely, from exp-time to poly-time. The operation, called matrixization, essentially builds a vector of length m^2 from a vector of length m by copying.

Surprisingly small attention has been paid to the models which achieve the settled exponential speedup. (The poly-space to poly-time speedup needs the unproved assumption that PTIME$\neq$PSPACE.) The only model achieving the exponential speedup, to the author's best knowledge, is nondeterministic PRAMs in [3]. Our vector operation seems better in its simplicity and irreducibility than that model which includes unusually complicated communication facilities depending on the coexistence of nondeterminism and parallelism.

Two supplementary but not less important results in this paper are as follows. It is shown that the vector machines (VMs in short) can be simulated by PRAMs with virtually no time loss if each processor is equipped with the multiplication instruction that can only be used for computing addresses of the shared memory (division is not necessary). Thus we can say, for example, PRAMs with multiplication can recognize in poly-log time any set in PTIME. Secondly we claim that the matrixization operation is more practical than the shared memory communication of PRAMs. To do so, we will give a subclass of the VMs hardware realization of which is clearly more feasible than PRAMs of the same size. They are less powerful in general than PRAMs but still show the same performance for a number of common specific problems such as computing connected components and minimum spanning trees of graphs.

Finally it should be noted that the basic idea of the existing "super computers" is far from the idea of PRAMs but is somehow close to that of VMs in the broad sense. This paper will hopefully activate research on VMs which the author believes are the most promising candidate of truly parallel computers.

2. Vector Machine Architectures.

A *vector of length* m is denoted by $(a_1,a_2,\ldots,a_m)$ where each a_i is a *scalar* (nonnegative integer). Let $A=(a_1,a_2,\ldots,a_i)$, $B=(b_1,b_2,\ldots,b_j)$ and $\cdot$ be a binary operation for scalars. Then $A\cdot B$ is defined as $(a_1\cdot b_1,\ a_2\cdot b_2,\ \ldots\ ,a_k\cdot b_k)$ where $k=\min(i,j)$. $(a_1,a_2,\ \ldots\ ,a_i)^l=(a_1,a_2,\ \ldots\ ,a_i)$ if $l=1$ and $=((a_1,a_2,\ldots,a_i)^{l-1},a_1,a_2,\ldots,a_i)$ if $l>1$. Key vector operations in this paper are called *matrixization* and *dematrixization*. In general, the matrixization operation makes a vector of length m^2 from a vector $(a_1,a_2,\ldots,a_m)$ of length m as follows:

$$((a_1,\ldots,a_{m/c})^c,(a_{m/c+1},\ldots,a_{2m/c})^c,\ldots,(a_{(c-1)m/c+1},\ldots,a_m)^c)^{m/c}.$$

As for values of c, three different ones are enough, namely, $c=1$, $c=m$ and $c=\sqrt{m}$. Matrixization operations for $c=1$, $c=m$ and $c=\sqrt{m}$ are denoted by $\downarrow$, $\rightarrow$ and $\searrow$, respectively. (See Fig.1. Two dimensional expressions used there are just for easier understanding. All vectors in this paper are one-dimensional.) As a special case, $\downarrow(a_1)=\rightarrow(a_1)=\searrow(a_1)=(a_1,a_1)$. The dematrixization operation is described using function *COMMON*. It is defined as $COMMON(a_1,\ \ldots,\ a_m)=0$ if $a_1=a_2=\ \ldots\ =a_m=0$ and $=a$ if a is the only one positive value taken by nonnegative a_i (i.e., each $a_i=0$ or a). Application of *COMMON* to the values more than two of which take different positive values, is illegal. For a vector $A=(a_1,\ \ldots,a_m,\ a_{m+1},\ \ldots,\ a_{2m},\ \ldots,\ a_{m^2-m+1},\ \ldots,a_{m^2})$, $COMMON(A)$ is a vector of length m defined by $(COMMON(a_1,\ \ldots,a_m),\ COMMON(a_{m+1},\ \ldots,\ a_{2m}),\ \ldots,COMMON(a_{m^2-m+1},\ \ldots,a_{m^2}))$.

In the rest of the paper, a, b, c, ... are used for scalar constants, x, y,z, ... for scalar variables and X, Y, Z, ... for vector variables. $X:=Y$ denotes an assignment instruction where the length of X (denoted by $|X|$) is $|Y|$ if $|Y|\leqq|X|$ and $|X|$ otherwise (i.e., the excess portion of Y is cut off).

Now we are ready to give a VM architecture (a set of instructions) called *MATRIX-COMMON*, which includes the following instructions:

(i) Scalar instructions

$x:=y+z$
$x:=y\dot{-}z$ ($y\dot{-}z=y-z$ if $y\geqq z$ and $=0$ otherwise)
$x:=y\div z$ ($y\div z=y/z$ if $y\geqq z$ and $=y$ otherwise)
If $x>0$ go to label
halt

(ii) Scalar-vector instructions

$y:=X$ (y holds the value of the first element of X)
$X:=y$ ($|X|=1$ whatever the length of X before execution is)

(iii) Vector instructions

$X:=Y+Z$, $X:=Y\dot{-}Z$, $X:=Y\div Z$
$X:=\downarrow Y$, $X:=\rightarrow Y$, $X:=\searrow Y$
$X:=COMMON(Y)$

Any of those instructions is executed in a single step. Note that the length of each vector variable X at some moment is determined by the most recent execution of the assignment instruction on the left-hand side of which X appears. We can use a special constant vector $I_0=(1,2,$

3, 4, ...). Its length is assumed to be longer than any other vector existing at that moment. Also two special vectors called *input* and *output vectors* are available.

Theorem 1. Let $f(n)$ be n^c or $\log^c n$ for a constant c. Then if a set $L \subseteq \{0,1\}^*$ is recognized in time $2^{f(n)}$ by one-tape TMs then L is recognized in time $O(f^2(n))$ by MATRIX-COMMON VMs.

Theorem 2. If a set L is recognized in time $T(n)$ ($T(n) \geqq \log n$) by MATRIX-COMMON then L is recognized in time $O(T(n))$ by PRAMs with unit-cost multiplication.

Thus multiplication in PRAMs is really powerful just as it is so in sequential RAMs [5]. We should be very careful in introducing multiplication to PRAMs although it is not likely that the operation still keeps its essential power when the number of processors is restricted to, say, n^2. The PRAMs in Theorem 2 do not need $\dot{-}$ or $\div$ but the usual subtract operation is enough. Also the multiplication operation needs to be used only for computing addresses of the shared memory. The fork instruction in [3] does not fit this theorem. All processors must be active from the beginning of the execution.

Now we introduce a subclass of MATRIX-COMMON. 2DIM-MATRIX-COMMON is a VM architecture which is the same as MATRIX-COMMON but the following. (i) It does not include $\searrow$. (ii) All vectors appearing in instructions are of length n or n^2 where n is the length of the input vector. This architecture is motivated by the parallel model called mesh of busses [6]. Fig.2 shows its structure. Each $P_{i,j}$ is a RAM. Each COLUMN_j and ROW_i can be viewed as a single cell of the shared memory that can be replaced by a communication bus. Physical realisability seems to be very good but it is intuitively less powerful than usual PRAMs of n^2 processors because $P_{i,j}$ can access only ROW_i and COLUMN_j. Those merit and demerit are exactly the same in 2DIM-MATRIX-COMMON. From this viewpoint, the matrixization operation seems to be realistic rather than powerful. An interesting feature is that although the restriction it shows the same power as PRAMs for a couple of combinatorial problems.

Theorem 3. Connected components of graphs, minimum spanning trees of graphs and sorting are computed in time $O(\log n)$ on 2DIM-MATRIX-ARBITRARY, 2DIM-MATRIX-MAX and randamized 2DIM-MATRIX-COMMON, respectively. (ARBITRARY picks a positive value nondeterministically and MAX the maximum value. Those, including COMMON, correspond to the write-conflict resolution in PRAMs.)

3. Proof Techniques.

Algorithms in Theorem 3 are essentially the same as those for the mesh of busses. In PRAMs each processor can execute conditional branch instructions but not in VMs. Operations $\dot{-}$ and $\div$ play an important role to handle this problem. This problem also arises frequently in other theorems.

Theorem 1 is one of those theorems in which the proof goal is clear but the way to there needs a lot of tricky techniques. Operation $\searrow$ plays a key role throughout the proof. Among many others we shall present the following fundamental technique. Let $B_i(m)$ be the zero-one vector of length i

that represents binary number m. For example, $B_8(9)=(0,0,0,0,1,0,0,1)$. Suppose that the following vectors are given for some $i=2^j$.

$$A=(B_i(0)^a, B_i(1)^a, \ldots, B_i(2^i-1)^a), \quad a=2^i/i.$$

$$B=(B_i(0)^b, B_i(1)^b, \ldots, B_i(2^i-1)^b)^{2^{0.5i}}, \quad b=2^{1.5i}/i.$$

$$C=((1)^i, (0)^i)^c, \quad c=2^{2i}/2i.$$

$$D=((0)^i, (1)^i)^d, \quad d=2^{3i}/2i.$$

Then

$$\searrow(A \div C)+\searrow(B \div D)$$

gives us

$$(B_{2i}(0)^e, B_{2i}(1)^e, \ldots, B_{2i}(2^{2i}-1)^e), \quad e=2^{2i}/2i.$$

This means we can make all zero-one strings of length $2i$ in constant steps using those of length i and operation $\searrow$. Then it follows that we can get all zero-one strings of length, say, 2^{n^2} in $O(n^2)$ steps.

Theorem 2 is not so difficult. Note that we do not have to compute directly the square root function when simulating operation $\searrow$.

References.

[1] Chandra, A.K., Kozen, D.C., and Stockmeyer, L.J., *Alternation*, JACM 28,114-133 (1981).

[2] Chandra, A.K., Stockmeyer, L.J., and Vishkin, U., *A complexity theory for unbounded fan-in parallelism*, 23rd FOCS, 1-13 (1983).

[3] Fortune, S., and Willie, J., *Parallelism in random access machines*, 10th STOC, 114-118 (1978).

[4] Goldschlager, L.M., *A unified approach to models of synchronous parallel machines*, 10th STOC, 89-94 (1978).

[5] Hartmanis, J, and Simon, J. *On the power of multiplication in random access machines*, 15th SWAT, 13-23 (1974).

[6] Iwama, K., *Feasible but still powerful PRAMs*, Technical Report COMP86-53, IECEJ (1986).

[7] Pratt, V.R. and Stockmeyer, L.J., *A characterization of the power of vector machines*, JCSS 12, 198-221 (1976).

$A = (a_1, a_2, a_3, a_4)$

$\downarrow A = (a_1, a_2, a_3, a_4,\ a_1, a_2, a_3, a_4,\ a_1, a_2, a_3, a_4,\ a_1, a_2, a_3, a_4)$

$\rightarrow A = (a_1, a_1, a_1, a_1,\ a_2, a_2, a_2, a_2,\ a_3, a_3, a_3, a_3,\ a_4, a_4, a_4, a_4)$

$A = (a_1, a_2\ a_3, a_4)$

$\searrow A = (a_1, a_2, a_1, a_2,\ a_3, a_4, a_3, a_4,\ a_1, a_2, a_1, a_2,\ a_3, a_4, a_3, a_4)$

Fig. 1. Vector operations.

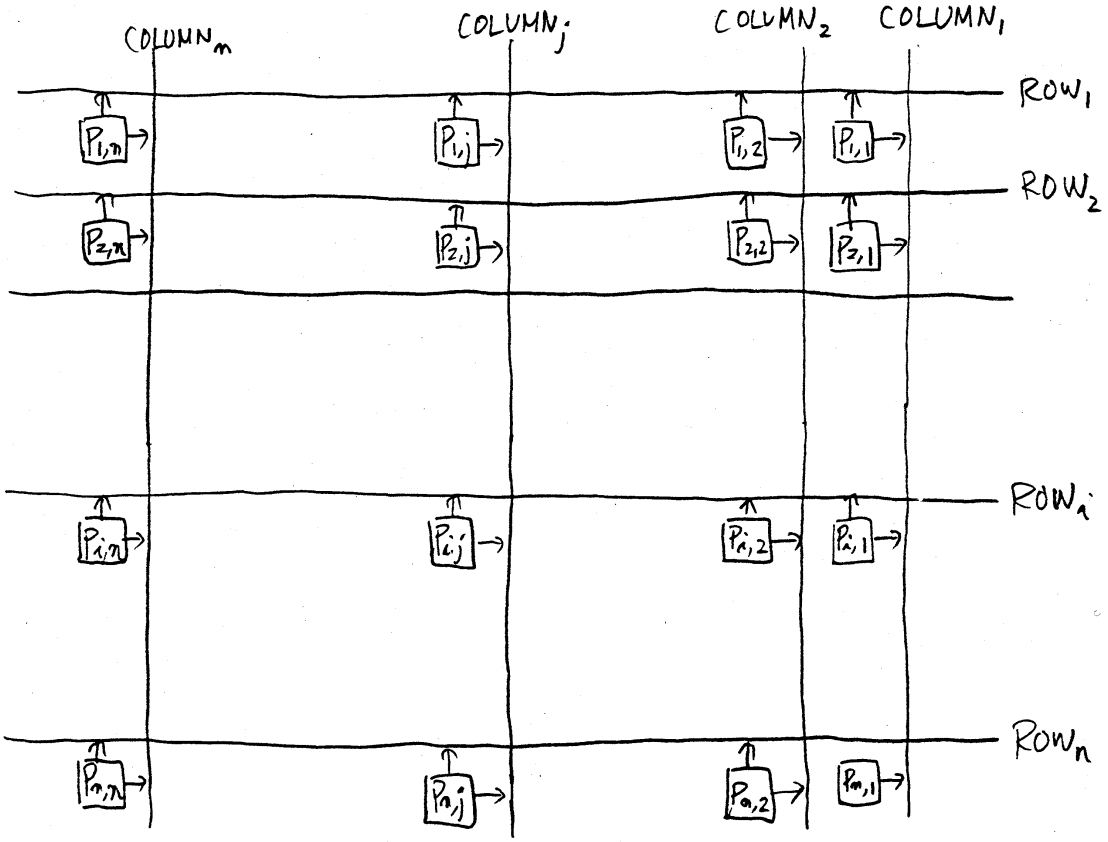

Fig. 2. Mesh of busses