Bull. Nov. Comp. Center, Comp. Science, 6 (1997), 59–69

Comparison of two models for associative parallel computations*

A.S. Nepomniaschaya, M.A. Vladyko

For fine-grained associative parallel systems of the SIMD type with vertical data processing we analyze two models of associative processing: the STAR-machine and the orthogonal machine. We have obtained that the STAR-machine simulates the orthogonal machine run in constant time while the orthogonal machine simulates the STAR-machine run in time which is proportional to the number of processing elements.

1. Introduction

The revived interest in the associative (content-addressable) architecture results from remarkable advances in the VLSI technology [1]. Of special interest is a class of associative parallel processors belonging to the fine-grained SIMD systems with bit-serial (vertical) processing and simple single-bit processing elements (PEs) [2]. In such systems input data are physically loaded in a matrix memory such that each data item occupies an individual row and is processed with its own processing element. These systems provide a massively parallel search by contents and processing of unordered data. They perform basic operations of searching such as exact match, greater than, greater or equal to, less than, less or equal to, maximum, minimum, greatest lower bound, least upper bound, between limits and outside limits which take time proportional to the number of bit columns in a field, but not to the number of data items being searched [3–6].

For employing the remarkable properties of an associative architecture it is essential to design associative parallel algorithms and simultaneously to define such a representation of input data which would make possible the solution of problems in a natural and robust way.

The main goal of this paper is to analyze two models performing the bit-serial data processing. Such an investigation is interesting owing to the following reasons. Firstly, by means of a formal model one can study new associative algorithms for different problem oriented shells. Secondly, as a result of analyzing associative algorithms there arise useful ideas for improving the associative architecture. Finally, the associative techniques can

*Partially supported by the Russian Foundation for Basic Research under Grant 96-01-01704.

be used on conventional sequential and parallel computers for improving the programming productivity [7]. Note that non-homogeneous high-performance computing systems generally include an associative parallel processor as a component for solving non-numerical problems [8].

Here, we compare the run of the STAR-machine [9] with the run of the orthogonal machine [10–11]. The STAR-machine is based on a STARAN-like associative parallel processor [5, 8]. The high-level language STAR is an extension of Pascal by adding new data types and the correponding operations for them for simulating the vertical data processing. The orthogonal machine includes main features of an associative computer of the SIMD type [12]. Both models are employed for analyzing associative algorithms [10–11, 13–17]. We have obtained that the STAR-machine simulates the orthogonal machine run in constant time while the orthogonal machine simulates the STAR-machine run in time which is proportional to the number of processing elements. We have also obtained a condition of belonging the STAR-machine to the class of vector machines introduced by Pratt and Stockmeyer [18].

2. The STAR-machine

The STAR-machine is defined as an abstract model of the SIMD type with vertical data processing. It consists of the following components:

- a sequential control unit where programs and scalar constants are stored;
- an associative processing unit consisting of m single-bit PEs;
- a matrix memory for the associative processing unit.

Its matrix memory consists of cells each storing one bit. Input binary data are loaded in the matrix memory in the form of two-dimensional tables in which each datum occupies an individual row. A row (word) or a column (slice) may be accessed equally easy.

The associative processing unit is represented as h vertical registers each consisting of m bits. Vertical registers can be regarded as a one column array. The STAR-machine runs as follows. The bit columns of the tabular data are stored in the registers which perform the necessary Boolean operations and record the search results.

To simulate data processing in the matrix memory three new data types **word**, **slice** and **table** are used. The types **slice** and **word** are employed for bit column access and bit row access, respectively, and the type **table** is used for defining the tabular data. Assume that any variable of the type **slice** consists of m components which belong to $\{\mathbf{0}, \mathbf{1}\}^*$.

*For simplicity let us call *slice* any variable of the type slice.

Consider operations, predicates and functions for slices.

Let X, Y be variables of the type **slice** and i, j be variables of the type **integer**. We define the following operations:

SET(Y) sets all components of Y to **1**;
CLR(Y) sets all components of Y to **0**;
$Y(i)$ selects the i-th component of Y;
FND(Y) returns the ordinal number i of the first component **1** of Y, $i \geq 0$;
STEP(Y) returns the same result as FND(Y) and then resets the first component **1**;
NUMB(Y) returns the number i of components **1** of Y, $i \geq 0$;
MASK($Y, i..j$) sets components **1** from the i-th through the j-th positions, inclusively, and components **0** in other positions of the slice Y ($1 \leq i < j \leq m$);
MASK1(Y, k) sets the alternation in Y consisting of k zeros and k ones, where $k = 2^i$, $i \geq 0$ (for example, MASK1($Y, 1$) denotes the alternation of the form **01**);
MIR(Y) returns the reverse of the contents of Y;
SHUFFLE(Y, k) performs the transpose of contents in each group from k ($k = 2^i$) components dividing them into two equal parts and placing components of the lower part upon the components of the upper one.

In the usual way we introduce the predicates ZERO(Y) and SOME(Y) and the following bitwise Boolean operations: X **and** Y is conjunction, X **or** Y is disjunction, **non** Y is negation, X **xor** Y is exclusive **or**.

We use the following standard functions:

Shift(Y, **down**, k) moves the contents of Y by k positions down, placing each component from the position N to the position $N + k$ ($N \geq 1$) and setting components **0** from the first through the k-th positions, inclusively. The function Shift(Y, **up**, k) is defined similarly.

Rotate(Y, **down**, k) performs a circular shift of the contents of Y by k positions down. This function is similar to Shift(Y, **down**, k) except that the components which are shifted out at one edge of the slice Y are shifted in at its opposite edge. The function Rotate(Y, **up**, k) is defined similarly.

Let w be a variable of the type **word** and T be a variable of the type **table**. We employ the following operations:

$w(i)$ returns the i-th component (bit) of w;
$\#w$ yields the length of w;
ROW(i, T) returns the i-th row of the matrix T ($1 \leq i \leq m$);
COL(i, T) returns the i-th column of the matrix T;
WITH(Y, T) attaches the contents of the slice Y to the left of the matrix T.

It should be noted that the current version of the language STAR does not include the operation PRESS(X, Y) because in [19] we have considered a specialized processor for realizing it.

3. The orthogonal machine

The orthogonal machine consists of the matrix memory and the processing unit. In the matrix memory there is a constant number of square matrices (or blocks) having m cells each storing one bit. It is assumed that the processing unit consists of three vertical registers X, Y and M of length m.

Let us examine the elementary operations of the orthogonal machine.

The operations COL(i, j) and ROW(i, j) select the i-th column and the i-th row in the j-th block of the matrix memory, respectively.

The operations $\langle 0 \rangle$ and $\langle 1, i \rangle$ are used for setting components **0** in a register and for setting the i-th component of a register to **1** and all other components to **0**, respectively.

The operations CL$(2^i)X$ and CR$(2^i)X$ perform the left cyclic shift and the right cyclic shift of the contents of X by 2^i positions, respectively.

The operations L(X) and R(X) set all the components of the register X to **0** except the position where in the register X there is the leftmost component **1** or the rightmost component **1**.

It should be noted that the left and the right operations arise in the case when each data item occupies an individual column and the input data are processed by rows.

Consider the following operations which are obtained by means of the elementary ones. As shown in [11], any of them takes constant time proportional to $\log l$.

The operation DM(l) defines a mask whose first l components are set to **1** and others are set to **0**.

The operations FL(X) and FR(X) fill in a register with components **1** up to the leftmost or the rightmost component **1** in the register X.

The operations SL$(l)Y$ and SR$(l)Y$ shift the contents of Y by l bit positions left and right, respectively, where l bit positions, which become free, are set to **0**.

The operations CSL$(l)X$ and CSR$(l)X$ perform left and right cyclic shifts of the contents of X by an arbitrary number l positions, respectively.

Statements and the bitwise Boolean operations for registers are introduced in the obvious way. Note that the iteration statement **for** employs the brackets **do** and **repeat**, whereas the statement **while** uses **do** and **od**.

Following [11], we assume that any elementary operation takes one unit of time. Therefore time complexity of an algorithm is measured by counting the number of all the elementary operations performed in the worst case.

4. Comparison of two models run

In this section we compare the STAR-machine run with the orthogonal machine run.

Any of these models consists of the matrix memory and the associative processing unit however, they differ in a set of instructions. Nevertheless, they employ the following common operations: the selection of the i-th row or the i-th column in the matrix memory, writing the components **0** in a register, the creation of a mask, the Boolean operations, the shift and the cyclic shift operations.

At first, we consider how the STAR-machine simulates the orthogonal machine operations. To this end it is essential to construct algorithms for simulating the operations $\mathsf{L}(X)$ and $\mathsf{R}(X)$ for setting all the components of a register to **0** except the leftmost or the rightmost component **1** in the given register X, and the operations $\mathsf{FL}(X)$ and $\mathsf{FR}(X)$ for filling in a register with components **1** up to the leftmost or the rightmost **1** in X.

It should be noted that algorithms for performing the right operations can be obtained from the corresponding algorithms for performing the left operations on the STAR-machine by means of the operation $\mathsf{MIR}(X)$ for reversing the contents of the given register X. Therefore we have only to show how the STAR-machine simulates the operations $\mathsf{L}(X)$ and $\mathsf{LF}(X)$.

The statement $Y := \mathsf{L}(X)$ is simulated as follows:

$$i := \mathsf{FND}(X);\ \mathsf{CLR}(Y);\ Y(i) := \mathbf{1}.$$

For simulating the assignment $Y := \mathsf{FL}(X)$ the following statements are employed:

$$i := \mathsf{FND}(X);\ \mathsf{MASK}(M, 1..i);\ Y := Y \text{ or } M.$$

As a result of performing these statements the first i components of the slice Y are set to **1** and others do not change.

So, we have obtained the following claim.

Claim 1. *Let an orthogonal machine run $T(m)$ steps, where the input is of the length m. Then the STAR-machine simulates the run of the orthogonal machine in $5T(m)$ steps.*

Now, we will study how the orthogonal machine simulates the STAR-machine operations.

Firstly, consider a group of the STAR-machine operations $\mathsf{SET}(Y)$, $Y(i)$ and $\mathsf{MASK}(M, i..j)$ which is simulated in constant time on the orthogonal machine.

It is obvious that the operation $\mathsf{SET}(Y)$ is obtained as the bitwise negation of a register consisting of components **0**.

Before simulating the operation MASK$(M, i..j)$ on the orthogonal machine recall that in this model a mask is filled in with components **1** beginning with the first component, while in the STAR-machine one can create a mask beginning with any position. So, we employ the following simple algorithm:

$$M := \mathsf{DM}(j);\ X := \mathsf{DM}(i-1);\ X := \mathsf{non}\ X;\ M := M\ \mathsf{and}\ X.$$

For simulating the statement $k := Y(i)$ on the orthogonal machine we create the register X whose i-th component coincides with the i-th component of the given register Y and others are equal to **0**:

$X := \langle 1, i\rangle$; $X := Y$ **and** X;
if ZERO(X) **then** $k := 0$ **else** $k := 1$.

Consider another group of the STAR-machine operations MASK1(Y, k), FND(Y), STEP(Y) and SHUFFLE(X, k) which is simulated on the orthogonal machine in time $O(\log m)$. Assume that $m = 2^n$ and $k = 2^i$.

The operation MASK1(Y, k) writes into the slice Y the alternation consisting of k zeros and k ones. It is simulated as follows:

$M :=$ DM(2^i);
for $j := n-1$ **downto** $i+1$ **do**
  $Y :=$ SR$(2^j)M$; $M := Y$ **or** M
repeat;
$Y :=$ SR$(2^i)M$.

Let us explain this algorithm. Initially the first k components of the mask M are set to **1**. At any j-th iteration the result of shifting the contents of the mask M by 2^j positions right is stored in the register Y. After performing the statement $M := Y$ **or** M the number of groups consisting of k ones increases. One can check that after ending the loop in the register Y there is an alternation of k ones and k zeros. Therefore for obtaining the required alternation it is essential to perform the last statement.

For simulating the statement $k :=$ FND(Y) we employ the following algorithm:

if ZERO(Y) **then** $k := 0$ **else** $k := 1$;
$M :=$ L(Y);
for $j := n-1$ **downto** 0 **do**
  $X :=$ SL$(2^j)M$;
  if SOME(X) **then**
    begin $k := k + 2^j$; $M := X$
    end;
repeat.

Explain the main idea of this algorithm. At first the position of the leftmost component **1** of the register Y is saved in the register M. At any j-th iteration the contents of the register M is shifted by 2^j positions to the left and the shift result is written in the register X. If the register X consists of components **0**, then at the next iteration the contents of the register M will be shifted half its current shift length.

Remark 1. For simulating the statement $k := \mathsf{FND}(Y)$ one can employ the following simple algorithm. First we perform the statement $M := \mathsf{L}(Y)$. Then we calculate the number of the left shifts when the register M does not completely consist of components **0**. However, such an algorithm takes $O(m)$ time.

Since the operations $\mathsf{FND}(Y)$ and $\mathsf{STEP}(Y)$ yield the same result, for simulating the operation $\mathsf{STEP}(Y)$ it is sufficient to show how to replace the leftmost component **1** of the register Y with the component **0**. To this end we employ the following simple algorithm:

$$X := Y;\ X := \mathsf{L}(X);\ X := \mathsf{non}\ X;\ Y := X\ \mathsf{and}\ Y.$$

For simulating the operation $\mathsf{SHUFFLE}(X, k)$ we will utilize the operation $\mathsf{MASK1}(Y, k)$, where $k = 2^i$.

$M := \mathsf{DM}(2^{i-1})$;
for $j := n - 1$ **downto** i **do**
  $Y := \mathsf{SR}(2^j)M$; $M := Y$ **or** M
repeat;
$Y := \mathsf{SR}(2^{i-1})M$;
/* We have performed the operation $\mathsf{MASK1}(Y, 2^{i-1})$. */
$M := \mathsf{SR}(2^{i-1})X$; $M := Y$ **and** M;
/* For all the groups we save their left parts in the register M. */
$X := X$ **and** Y; $Y := \mathsf{SL}(2^{i-1})X$;
/* For all the groups we save their right parts in the register Y. */
$X := Y$ **or** M.

It is not difficult to understand that the orthogonal machine simulates the operations $\mathsf{MASK1}(Y, k)$ and $\mathsf{SHUFFLE}(X, k)$ in the same $O(\log m)$ time.

Now, let us explain the algorithm for simulating the operation $\mathsf{MIR}(X)$ of reversing the contents of the register X. First we perform the left cyclic shift by using the statement $X := \mathsf{CL}(2^{n-1})X$. Then, we execute the operation $\mathsf{SHUFFLE}(X, 2^j)$ $n - 1$ times for $j = n - 1, \ldots, 1$. Since the orthogonal machine simulates this operation in $O(\log m)$ time, it will simulate the operation $\mathsf{MIR}(X)$ in $O(\log^2 m)$ time.

Remark 2. One can simulate the statement $Y := \mathsf{MIR}(X)$ by means of the following simple algorithm which requires $O(m)$ time. At first we set all the

components of the register Y to **0**. Then we perform m times a loop with respect to the parameter i which includes the following steps:

- to create a register (mask) consisting of components **0** except its first component;
- to perform the cyclic shift to the right by one bit position of the contents of the register X and to write down the shift result into the register X;
- to mask the first component of the register X being the current component for reversing the initial register;
- to add to the register Y the position of the selected component after its shift to the right by $i-1$ bit positions.

Finally, consider the following simple algorithm for simulating the statement $k := \mathsf{NUMB}(Y)$ on the orthogonal machine.

```
    k := 0; M := Y;
/* In the register M there is a copy of the register Y. */
    while SOME(M) do
      k := k + 1; X := L(M);
      X := non X; M := M and X;
    od.
```

Here, the value of k is increased when the left component **1** is deleted from M. This algorithm requires $O(m)$ time since it is essential to analyze all the components of the register Y.

Remark 3. Correctness of the algorithm for simulating $\mathsf{NUMB}(Y)$ is verified by induction on the number of components **1** in the slice Y. Correctness of other algorithms is checked by induction on n.

Claim 2. *Let a STAR-machine run $T(m)$ steps, where the input is of the length m. Then the orthogonal machine simulates the STAR-machine run in $O(mT(m))$ time.*

Claim 3. *Let a STAR-machine run $T(m)$ steps and do not use the operation $\mathsf{NUMB}(Y)$. Then the orthogonal machine simulates the STAR-machine run in $O(T(m)\log^2 m)$ time.*

In [18], Pratt and Stockmeyer have defined a class of vector machines for which $P = NP$, that is, the sets accepted in polynomial time by a non-deterministic vector machine can be accepted in polynomial time by a deterministic vector machine. In [10], Sykora and Otrubova have proved

that vector machines and orthogonal machines are from the same machine class.

From Claims 1 and 3 we obtain the following

Corollary. *Let* $T(m) \geq \log m$. *Then the STAR-machine belongs to the class of vector machines, if in the STAR-machine the operation* NUMB(Y) *is forbidden.*

5. Conclusions

In this paper we have compared the run of two models. We have shown that the STAR-machine simulates the orthogonal machine run in constant time, while the orthogonal machine simulates the STAR-machine run in time which is proportional to the number of processing elements. However, if the use of the operation NUMB(Y) is forbidden, the STAR-machine simulates the orthogonal machine run in $O(T(m) \log^2 m)$ time. Since for $T(m) \geq \log m$ the orthogonal machine belongs to the class of vector machines, we have obtained a condition of pertaining the STAR-machine to this class.

From analyzing different associative algorithms written in the language STAR we come to the following conclusions:

- the operations FND(Y), STEP(Y) and $Y(i)$ are very convenient and a lot of STAR procedures employ them;
- if the operation NUMB(Y) is forbidden, one can easily write down a procedure which returns the result of this operation;
- along with the operations FND(Y) and STEP(Y) it is useful to have at hand the operations for defining the position number of the *last* component **1** in the slice Y and for replacing this component with **0**.

Note that in the latter conclusion we have an analogy with the left and the right operations in the orthogonal machine. Such basic operations can be applied, for example, for representing Falkoff's algorithm for finding the nearest lower bound or the least upper bound on associative parallel processors.

Acknowledgements. We would like to thank Prof. Sykora for setting the problem.

References

[1] K.E. Grosspietsch, *Associative Processors and Memories*, A survey, IEEE, Micro, June, 1992, 12–19.

[2] Y.I. Fet, *Vertical Processing Systems*, A survey, IEEE, Micro, February, 1995, 65–75.

[3] A.D. Falkoff, *Algorithms for parallel-search memories*, J. of the ACM, **9**, No. 10, 1962, 448–510.

[4] J.L. Potter, *Associative Computing: A Programming Paradigm for Massively Parallel Computers*, Kent State University, Plenum Press, New York and London, 1992.

[5] C.C. Foster, *Content Addressable Parallel Processors*, Van Nostrand Reinhold Company, New York, 1976.

[6] A.S. Nepomniaschaya, *Investigation of Associative Search Algorithms in Vertical Processing Systems*, Proc. of the Intern. Conf. "Parallel Computing Technologies", Obninsk, Russia, 1993, 631–641.

[7] J. Potter, J. Baker, A. Bansal, S. Scott, C. Leangsuksun, C. Asthagiri, *ASC – an associative computing paradigm*, Computer: Special Issue on Associative Processing, **27**, No. 11, 1994, 19–25.

[8] N. Mirenkov, *The Siberian approach for an open-system high-performance computing architecture*, Computing and Control Engineering Journal, **3**, No. 3, May, 1992, 137–142.

[9] A.S. Nepomniaschaya, *Language STAR for Associative and Parallel Computation with Vertical Data Processing*, Proc. of the Intern. Conf. "Parallel Computing Technologies", World Scientific, Singapore, 1991, 258–265.

[10] B. Otrubova, O. Sykora, *Orthogonal Computer and its Application to Some Graph Problems*, Parcella'86, Academie Verlag, Berlin, 1986, 259–266.

[11] A. Huebler, O. Sykora, *Image processing and recognition algorithms for an orthogonal computer*, Computers and Artificial Intelligence, **6**, No. 2, 1987, 131–149.

[12] K. Richter, *A Parallel Computer System SIMD*, Artificial Intelligence and Information–Control Systems of Robots, North–Holland Publ. Comp., Bratislava–Smolenice, 1984, 309–319.

[13] A.S. Nepomniaschaya, *Comparison of two MST algorithms for associative parallel processors*, Proc. of the 3-d Intern. Conf. "Parallel Computing Technologies", PaCT–95, St. Petersburg, Russia, Lecture Notes in Computer Science, Springer–Verlag, Berlin, **964**, 1995, 85–93.

[14] A.S. Nepomniaschaya, *Representations of the Prim–Dijkstra Algorithm on Associative Parallel Processors*, Proc. of VII Intern. Workshop on Parallel Processing by Cellular Automata and Arrays. Parcella'96, Academie Verlag, Berlin, 1996, 184–194.

[15] A.S. Nepomniaschaya, *Representation of the Gabow Algorithm for finding smallest spanning trees with a degree constraint on associative parallel processors*, Euro–Par'96 Parallel Processing. Second Intern. Euro–Par Conf. Lyon, France, August, 1996. Proceedings, Lect. Notes in Comp. Sci., Springer–Verlag, Berlin, 1996, **1**, No. 1123, 813–817.

[16] A.S. Nepomniaschaya, *An associative version of the Prim–Dijkstra algorithm and its application to some graph problems*, Andrei Ershov Second Intern. Memorial Conf. "Perspectives of System Informatics", Lecture Notes in Computer Science, **1181**, Springer–Verlag, Berlin, 1996, 203–213.

[17] J. Miklosko, R. Klette, M. Vajtersic, J. Vrto, *Fast Algorithms and their Implementation on Specialized Parallel Computers*, Special Topics in Supercomputing, North–Holand, **5**, 1989.

[18] V.R. Pratt, L.J. Stockmeyer, *A characterization of the power of vector machines*, Journal of Computer and System Sciences, **12**, No. 2, 1976, 198–221.

[19] A.S. Nepomniaschaya, Y.I. Fet, *Investigation of some hardware accelerators for relational algebra operations*, The First Aizu Intern. Symposium on Parallel Algorithms/Architecture Synthesis, IEEE Computer Society Press, Aizu, Japan, 1995, 308–314.