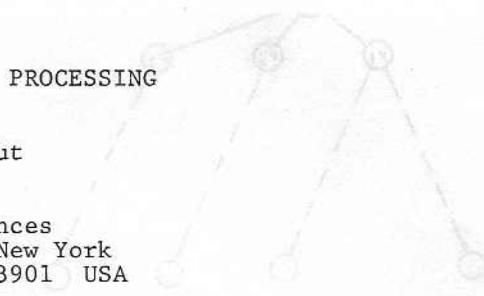
Proc. 23rd Symposium on Foundations of Computer Science, IEEE, 1982, pp. 272-279



USING CLERKS IN PARALLEL PROCESSING

Quentin F. Stout

Mathematical Sciences
State University of New York
Binghamton, New York 13901 USA

ABSTRACT

Some models of parallel computation consist of copies of a single finite automaton, connected together in a regular fashion. In such computers clerks can be a useful data structure, enabling one to simulate a more powerful computer for which optimal algorithms are easier to design. Clerks are used here to give optimal algorithms for the 3-dimensional connected 1s problem on a parallel processing array, and a circle construction problem on a pyramid cellular automaton.

1. INTRODUCTION

In this paper we solve two open problems in the area of parallel processing. However, we consider the method of solution to be of more interest than the problems themselves, even though one has been called a "classic open problem" of computer science. (Kosaraju[8]). Basically, we solve problems by designing algorithms for more powerful models of parallel computation, and then use a data structure we call a clerk to simulate the more powerful computer on the target model. This is a systematic method for attacking a wide variety of problems, and to illustrate this we have chosen two considerably different open problems on two different models. The first appearance of clerks is in Stout[12], which solves yet another problem. This paper should be considered as an account of work still in progress.

The models we use are usually described in terms of finite state automata, but for our purposes it is easiest to take the equivalent view that each processor is a RAM with a fixed number of memory cells, each of which has a fixed length. These processors are called finite RAMs. Our first problem is for a pyramid cellular automaton (PCA), which has appeared in [4,11,12,14] and elsewhere. Given a fixed finite RAM we place a copy of it at each node of a complete 4-ary tree of height h, where h is a nonnegative integer. A processor at height k can send, in unit time, a fixed amount of information to any of its nine neighbors: four sons at height k-1, a father at height k+1, or the four adjacent processors at height k. (The base is at height 0, and processors on the edges think they are connected to outside processors in a special "edge" state. From this a processor can determine, for example, if it is part of the base. See [4].) All nine communication links can be used simultaneously independent of each other. An input to a PCA consists of loading desired values into the memory of the base processors, setting all processors except the apex into a quiescent state, and putting the apex into a start state. We pick coordinates for the processors in the base so that, when viewed from the apex, the processor in the lower left is at position (0,0) and the processor in the upper right is (n-1,n-1). Notice n=2**h.

> **Circle Problem:** Suppose initially one processor in the base of a PCA is labeled "A", another is labeled "B", and all others are labeled "unmarked". The problem is to label with an "A" all base processors whose distance to the "B" processor is no greater than the distance between the original "A" and "B" processors. The distance between the processors at (a,b) and (c,d) is sqrt[$(a-c)^2 + (b-d)^2$].

The circle problem is due to Sakoda[11], though we have altered it slightly to improve exposition.

Our second problem is for a mesh-connected computer. A 3-dimensional parallel processing array (3-PPA) [8] of size n consists of n^3 copies of a finite RAM, located at points with coordinates (a,b,c) where a, b, and c are integers and $0 \le a,b,c \le n-1$. Processors at (a,b,c) and (d,e,f) can exchange information in unit

0272-5428/82/0000/0272$00.75 © 1982 IEEE

time if and only if $1 = |a-d| + |b-e| + |c-f|$. Processors along the sides and edges can be thought of as being connected to outside processors which are in a special "edge" state. An input to a 3-PPA consists of setting the memory of all processors and simultaneously starting all of them in a special "start" state. In our application each processor is initially given either a 0 or a 1, which collectively represent a digitized 3-dimensional image. Two 1s are adjacent if they are in processors which share a side, and two 1s are connected if there is a path of adjacent 1s which goes from one to the other.

> **Connected 1s Problem:** Given an nxnxn input of 0s and 1s, can a 3-PPA of size n decide in O(n) time whether or not all the 1s are connected?

This "classic" open problem is due to Beyer[3], and has also appeared in [1,8,10]. The 2-dimensional problem was solved in [3,9], and here we show that the answer to the 3-dimensional problem, and in fact all higher dimensions, is "yes".

2. SOLUTION OF THE CIRCLE PROBLEM

Initially suppose that we are given a pyramid computer where each processor is a RAM of unlimited memory, able to do addition, multiplication, etc. in unit time, and where each base processor has been initialized to contain its coordinates. For this model we give an easy O(log(n)) time solution, and then we simulate the solution using clerks. The apex sends down a message to start, and the processor containing the "A" sends its coordinates up to the apex. The apex sends this back down to all processors, and then repeats the process for the "B" processor. Each processor calculates the square of the distance between the A and B, and then calculates the square of the distance from it to the B. If the second number is no greater than the first then it becomes an "A" processor, and otherwise is "unmarked". The total time is O(log(n)). Notice that each processor uses only a fixed number of registers, and the largest number ever created is $(2n-2)^2$, which needs 2*lg(n)+3 bits to be stored as a signed binary number. (lg is log base 2.)

We now show how to use clerks to simulate the above algorithm on a PCA. Since each processor is a finite RAM whose memory size is fixed and independant of n it is no longer possible for each base processor to store its position, nor to do calculations which depend on n. This is accomplished by the clerks, which in effect do all the work. Each clerk lies in a single column of the base and consists of L consecutive processors, where

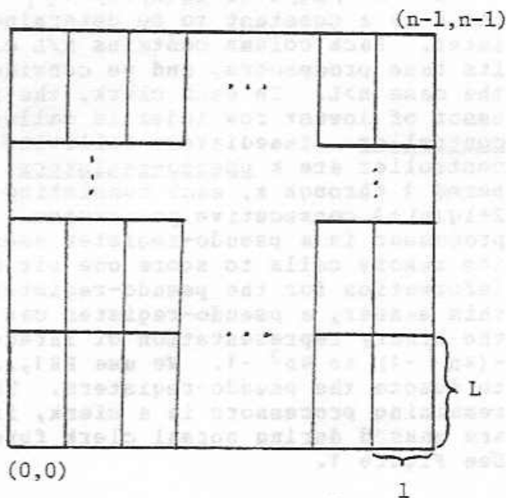
L = 2**⌈lg⌈k*(2*lg(n)+3)+1⌉⌉

and k is a constant to be determined later. Each column contains n/L clerks in its base processors, and we consider only the case n>L. In each clerk, the processor of lowest row index is called the controller. Immediately following the controller are k pseudo-registers, numbered 1 through k, each consisting of 2*lg(n)+3 consecutive processors. Each processor in a pseudo-register uses one of its memory cells to store one bit of information for the pseudo-register. In this manner, a pseudo-register can store the binary representation of integers from $-(4n^2-1)$ to $4n^2-1$. We use PR1,...,PRk to denote the pseudo-registers. The remaining processors in a clerk, if any, are unused during normal clerk functions. See Figure 1.

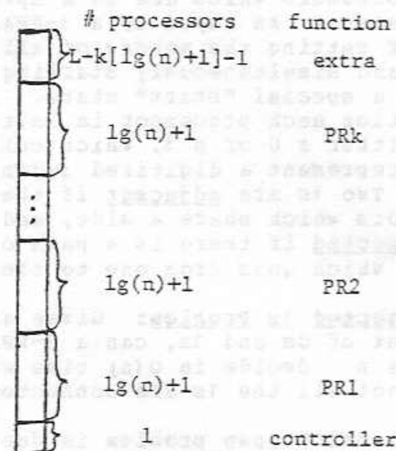
Each processor uses one memory cell to remember its type. There are a fixed number of types, representing the fact that the processor is a controller, the apex, part of PR1, etc. Once the type of a processor has been set its' behavior is determined throughout the algorithm. Processors at heights 1 through lg(L) are of type supervisor, and processors above these are of type boss. (The apex is a special type.) In each column each supervisor is above only one clerk or part of one clerk, but each boss is above two or more. When information is being sent from the apex to the clerks, each boss passes it to all of its sons, while each supervisor passes it only to its lower two sons. By doing this, all information arrives at a clerk only through the controller.

2.1 EXECUTING INSTRUCTIONS

Each clerk has a fixed set of instructions. For the circle problem these include copying, comparisons, addition, subtraction, multiplication, division, and square root extraction. To see how these are executed, suppose the next instruction is to copy PR2 to PR4. The controller passes this instruction to its neighbor, which passes it to its neighbor, and so on. When the first processor in PR2 receives it, it passes it along, sends a copy of the bit it is storing, and then sends an end-of-message (EOM) indicator. (The first processor in PR2 knows it is first since its type is PR1 and it is adjacent to a processor of type PR1.) This message is passed along, and as each processor in PR2 receives the EOM it adds its bit and then sends the EOM. When the stream reaches the first processor in PR4, it passes along the instruction, and then



a) Clerks on the base



b) Processors within a clerk

Figure 1. Clerk Organization in a PCA

stores the next bit received (without passing it along). Each processor in PR4 acts similarly, later processors having to wait longer between the arrival of the instruction and their bit. We call this process rolling a number into a pseudo-register. When the last processor in PR4 receives EOM it sends back a signal that the instruction is completed. The controller receives this in O(log(n)) time from the start of execution.

Comparisons and additions are similarly performed, with other arithmetic operations being somewhat more complicated. A suitable multiplication algorithm was given by Atrubin[2], and is repeated in Knuth[7,p.276]. For division and square root extraction one can either adapt the fast iterative algorithms, as in [7], or the on-line algorithms in [5,14]. (The on-line algorithms produce an answer using the digits {-1,0,1}, so their answer must be converted to standard binary notation.) Details of these operations will appear in Stout[12], and each has the property that it is completed in O(log(n)) time.

The crucial feature is that programs consist of a fixed finite number of instructions, which can therefore be stored in a finite RAM. Each instruction takes O(log(n)) time, but since the controller just issues an instruction and waits until it receives a signal to proceed, this dependence on n does not prevent it from being an automaton.

2.2 TYPING AND INITIALIZATION

Algorithms have three parts: typing each processor, initializing the pseudo-registers, and performing the calculations. Each part takes O(log(n)) time, giving O(log(n)) total time. The details of typing and initialization are similar to those in Stout[13], so we will only briefly sketch the process. First we identify supervisors and bosses, which requires us to determine lg(L). The apex states a message of 3 1s, one at a time, followed by EOM. Each processor receiving this passes it on to its lower left son, and whenever a processor receives the EOM it adds 2k 1s before sending the EOM along. When this reaches the base (at processor (0,0)) this processor passes it along to processor (0,1), which starts rolling the 1s onto the first row. When processor (0,0) receives the EOM it adds its 2k 1s and then the EOM. The last processor to have a 1 rolled onto it is at (0,k*(2*lg(n)+3)). This is the only processor not having other 1s pass over, and when it receives the EOM it starts it upwards. The first processor to receive the EOM which also helped send the original message downwards is at height lg(L). This is the highest level of a supervisor,

and it is straightforward to use this processor to set the type of all other supervisors and bosses in O(log(n)) time.

Since the bosses and supervisors have been set, from now on, unless otherwise specified, all messages from the apex will enter a clerk only through the controller, and all messages are sent to all clerks. First the apex sends a 4, which when received by a base processor tells it that it is of type controller. To identify the processors in each clerk's PR1, the apex sends three 5's followed by EOM. This is passed on, each processor adding two 5's when the EOM is received. When it reaches the controller it is passed to the next processor and rolled onto the clerk, and when the controller receives EOM two 5's are added and a message is sent upward to start the next step. Each previously untyped processor receiving a 5 is of type PR1. Each pseudo-register is built in similar fashion, finishing the typing of all processors in O(log(n)) time.

We initialize pseudo-registers so that each clerk contains the x and y coordinates of its controller, the x and y coordinates of the initial A and B processors, and L. These are similar, so we describe giving each clerk the x coordinates of its controller. The apex first sends a 0 sign bit to all children, then a 0 to its two left children and a 1 to its two right children, simultaneously, followed by EOM. Each boss and supervisor passes this along, adding a final 0 for left children and a 1 for right ones. Obtaining the x and y coordinates of A and B requires having them initiate the stream, with each processor adding initial bits depending on which son is passing up the message. When this reaches the apex it is sent back down to all clerks.

2.3 CALCULATING

We complete the algorithm by doing calculations within clerks. Let (xc,yc) be the coordinates of the controller, (xA,yA) be the coordinates of the initial A processor, and (xB,yB) the coordinates of the initial B processor. Each clerk first computes $(xA-xB)^2 + (yA-yB)^2$ (i.e., the square of the distance from A to B). Let d represent this value.

Within a clerk, the processors to be labeled "A" form an (perhaps empty) interval. There are three possibilities: $yc \geq yB$, in which case the interval, if any, goes from the controller upwards; $yc+L-1 \leq yB$, where the interval goes downward from the other end of the clerk; or $yc<yB<yc+L-1$, in which case the interval is in the middle of the clerk. In O(log(n)) time each clerk determines which case applies to it. These are similar, so we discuss only what happens in the first one. The clerk tests

$$(xc-xB)^2 + (yc-yB)^2 > d ,$$

and if true then no processors are labeled A. If false, it tests

$$(xc-xB)^2 + ((yc+L-1)-yB)^2 \leq d ,$$

and if this is true then all processors in the clerk become A's. Otherwise it determines the largest j in $0 \leq j < L-1$ such that

$$(xc-xB)^2 + ((yc+j)-yB)^2 \leq d ,$$

for it is the controller and the next j processors which become A. It solves this quadratic equation by using, among other operations, division and square root extraction. The clerk converts the j into unary notation (using the entire clerk) and labels the controller and the next j processors. Finally, the clerk signals that it is finished. Whenever the lower two sons of a supervisor, or all four sons of a boss, signal that they are done, that processor in turn tells its parent that it is done. The entire algorithm is finished when the four sons of the apex are done. The last detail is that one must inspect the computations to determine how many pseudo-registers are needed, and chose k to be this value.

Theorem There is an automaton such that any pyramid cellular automaton constructed from this automaton will solve the circle problem in time linear in its height. □

3. SOLUTION OF THE CONNECTED 1S PROBLEM

Our solution to the connected 1s problem is based on a divide-and-conquer strategy. This was used by Kosaraju[8] in his work on an extension of the 2-dimensional connected 1s problem, and by Nassimi and Sahni[10] for labeling connected components of 1s using mesh-connected computers of arbitrary dimension. The Nassimi and Sahni mesh-connected computers had a wordsize of O(log(n)), and it is basically their algorithm we will simulate. Their algorithm labels each processor containing a 1, with two processors having the same label if and only if their 1s are connected. Each processor containing a 1 starts with a label consisting of a concatenation of its coordinates, and when finished each component is labeled with the minimium such label among its processors.

The algorithm labels components within small cubes, and then combines these to label components within larger ones. What appear to be seperate components in a small cube may be connected together in a larger one, but only if they touch a side of the smaller cube. When 8 cubes of

edgesize k combine to form one of edgesize 2k, one need only determine changes in component labels due to adjacent 1s in seperate cubes, and then inform all processors on the sides of the resulting cube. This is repeated until the sides of the entire computer are correctly labeled, and then it is subdivided into 8 subcubes and the process is repeated. Smaller cubes are repeatedly merged to correctly label the sides of the target cube, which is then divided into 8 smaller target cubes.

The crucial part is the joining of 8 cubes of edgesize k to form one of edgesize 2k. To simplify discussion we call the original 8 cubes "brothers" and the resulting cube the "parent". Each processor containing a 1 which lies on a side of a brother which is adjacent to another brother checks the adjacent processor in the other brother. If it also contains a 1 then the processor forms a record containing its label, the label of the adjacent processor, and its coordinates. Meanwhile, processors on the sides of the parent form a record containing their label and coordinates. This results in $O(k^2)$ records to be brought together to decide the new labelings. In $O(k)$ time these are moved to a subcube of edgesize $O(k^{2/3})$. There the new labels are determined using a connected component algorithm of Hirschberg[6] for use on a paracomputer. Hirschberg's algorithm takes $O(\log(k)^2)$ time, but to simulate a single step of the paracomputer takes $O(k^{2/3})$ time on the subcube, resulting in $O(\log(k)^2 * k^{2/3})$ time to determine new labels. These are then sent back to the original processors, taking $O(k)$ time to arrive. Repeated use of this procedure solves the labeling problem in $O(n)$ time.

To adapt this algorithm to a clerk-based solution of the connected 1s problem basically involves four areas. First we need to show how to build clerks in a 3-PPA. Second, the most important feature of the relabeling of the parent is that most of the work is done in the subcube. The subcube needs $O(k^2)$ clerks, which will take $O(\log(n) * k^2)$ space. As long as k is $O(\log(n)^3)$ there will be sufficient space, but for smaller cubes a completely different procedure must be used. The 3-PPA will be divided into small initial cubes, labeled by this different procedure, and then all larger cubes will be multiples of these initial ones. Third, we note that moving the records during the relabeling of a parent's sides is the part which takes $O(k)$ time. Therefore to avoid an $O(n*\log(n))$ algorithm we must perform this movement in $O(k)$ time also, not the $O(k*\log(n))$ which we get from blindly letting each step on the mesh-connected computer be simulated in $O(\log(n))$ time in the clerks. Fourth, we must use the resulting labeling to decide the connected 1s problem.

3.1 CLERKS IN A 3-PPA

The structure of clerks is as before, with a suitable number of pseudo-registers of length $O(\log(n))$. Again we let L denote the length of a clerk. For simplicity we assume $L<n$ and both n and L are powers of 2. One difference is that we now need three overlapping sets of clerks. There is a set parallel to the x-axis, having their controllers in the left, a set parallel to the y-axis with their controllers at the bottom, and a set parallel to the z-axis with their controllers in front. See Figure 2. We use three sets so that, given any processor on the face of an initial cube, there is a clerk containing the processor and perpendicular to the face. Further, the processor will either be the controller or the last processor of the clerk. (Initial cubes will have an edgesize which is a multiple of the clerk length.) We say that the clerk is <u>attached</u> to the processor. No clerk is attached to more than one processor. Since processors on edges and corners have a choice of clerks to be attached to, we arbitrarily decree that clerks parallel to the x-axis have first priority, followed by clerks parallel to the y-axis. Notice some clerks are not attached to any processor, and processors in the interior of the initial cubes have no attached clerks. This attachment is determined by the clerks prior to labeling the initial cubes, and clerks stay attached throughout the algorithm. Since all later cubes are multiples of initial cubes, for all cubes it is still true that each surface processor has an attached clerk.

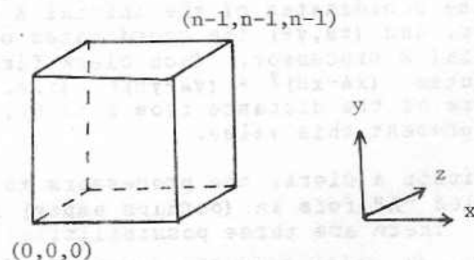


Figure 2. Orientation of a 3-PPA

Each processor is in three clerks, so we can think of each time unit as being divided in three parts, with a processor performing its action for each clerk during its part. Usually we will discuss clerks in only one direction, with the understanding that the other directions behave similarly.

Since we are allowed O(n) time to form the clerks it can be done in several straightforward ways. In O(n) time one can determine lg(n) and L, and from this set up a single clerk. If this clerk is, say, the one parallel to the x-axis with its controller at (0,0,0) then it can, in O(n) time, be copied along the bottom line. This line can then be copied along the bottom plane, and then the plane copied throughout the cube, in O(n) time. Clerks can also be initialized in O(n) time so that they contain the coordinates of their controller, their orientation (i.e., the axis they are parallel to), L, and lg(n). The details of all of this are left to the reader.

3.2 INITIAL CUBES

We need to determine the size of our initial cubes. We note that there is a C (which we will not determine) such that if a cube has edgesize greater than $C*lg(n)^3$ then there is room to carry out the Hirschberg procedure. Define k to be $2**\lceil lg\lceil C*lg(n)^3 \rceil\rceil$. Our initial cubes will have an edgesize of k. Since L is also a power of 2, L evenly divides K. First each clerk determines whether or not it is attached to a processor. If so then the processor is on the surface of an initial cube, and the clerk informs it of the directions "outward" to the cube. All uninformed cubes must be in the interior of an initial cube. This "outward" information will enable us to keep some operations within a single initial cube.

Each processor containing a 1 starts in an "unused" state and may later become "used". We start going around the surface of the cube in increasing order of the concatenated coordinates. Each surface processor has a "turn" which takes a fixed number of steps, and it is notified when to start its turn by its attached clerk. If the processor contains a 0 then nothing happens during its turn, and similarly nothing happens if it contains a 1 but has been "used". Otherwise it contains a 1 which lies on an untraced component. It turns "on", and informs each processor which is not in an outward direction. Each processor containing a 1 which is informed that a neighbor is "on" turns itself "on" and passes along the message, never passing it in an outward direction. This can continue for no more than k time units, after which no more processors can turn on. All clerks have been counting, and when this time is reached they change any "on" processors within them to "used". Further, if the clerk is attached to a surface processor which was turned on in this turn, then its label becomes that of the processor whose turn it was. There are only $O(k^2)$ turns, taking $O(k^3)$ time each, resulting in a total of $O(k^5)$ time, or $O(\log(n)^{15})$. Now the surfaces of the initial cubes are correctly labeled, and all larger cubes are labeled via the Nassimi and Sahni algorithm.

3.3 RECORD MOVEMENT

Suppose 8 brother cubes of edgesize k are being merged and relabeled to form a parent cube. We need to show that in only O(k) time we can move the records generated at the surfaces down to the subcube where the Hirshberg algorithm is performed. (Note: we are using a 3-PPA to simulate a mesh-connected computer which is simulating a paracomputer!) The problem of moving the data to the subcube can be solved if we can solve the following problem: suppose we have a cube of edgesize m, where m is a multiple of L and $m^{2/3}$ is an integer. Suppose each clerk perpendicular to the leftmost face of the cube has a record. In O(m) time we must move these records, one per clerk, into the m^2 clerks parallel to the x-axis in the rectangular box in the lower left corner with height $m^{2/3}$, length $L*m^{2/3}$, and depth $m^{2/3}$.

To solve this new problem, imagine the m^2 processors on the end as being in $m^{2/3}$ squares with sides $m^{2/3}$. These squares are numbered in some easily computible manner, from 1 to $m^{2/3}$. Each clerk parallel to the x-axis computes the number of the square on the line of that clerk, and each clerk determines if it is that clerk along this line. That is, if the square is number 4, it determines whether or not it is the fourth clerk from the left in this cube. Then the records are passed right until they reach the appropriate clerk along their line. What is happening is that entire squares are moving in parallel, finding their appropriate x position. This takes $O(L*m^{2/3})$ time, and then all squares are moved along the z direction until they are at the front of the cube. This takes O(m) time, and then they are moved down in the y direction until they are at the bottom. This also has taken O(m) time, and the movement is completed. Further, the inverse operations can be similarly performed in O(m) time, and we therefore conclude that we

can merge 8 brothers of edgesize k in O(k) time.

3.4 SOLVING THE CONNECTED 1S PROBLEM

The final details involve adapting the labeling algorithm into a solution of the connected 1s problem. The processor at (0,0,0) will be the one which determines the answer. When we set up the clerks we also tell each processor which way to send information towards the origin. We notice that for each labeled component of 1s there is exactly one processor whose concatenated coordinates equals the label of the component. When the labeling is completed, each clerk which is attached to a processor determines whether or not this condition is true, and if true starts a message toward the origin. If two such component messages arrive at a processor at the same time then it sends a "no" message toward the origin, and if the origin ever receives two such messages its answer is no.

Unfortunately, the labeling process will miss small components which lie entirely in the interior of an initial cube. To check for this, each processor at the lower left front corner of an initial cube starts a signal which systematically moves through the cube. (Remember that processors on the sides of this cube know which directions are outward.) It the signal reaches the upper back right corner it returns to where it started, but if it reaches an "unused" 1 it turns that 1 on, which then starts tracing out its component. Meanwhile the clerk attached to the processor in the lower front left of the cube is counting, and if the signal has not returned after a time equal to twice the number of processors in the cube then it knows that a new component has been found and sends a signal to the origin. In this case it also starts a second signal through the cube, just like the first, which either reaches the far corner and returns or else encounters an unused 1, representing yet another component. Again the one clerk is counting, and if the second signal also fails to return then it starts a "no" signal towards the origin. Meanwhile the clerk attached to the origin determines the maximum time it must wait. If two or more component signals, or any "no" signals, reach the origin within this time then the answer is no, while otherwise it is yes. This solves the 3-dimensional connected 1s problem in O(n) time, and the solution method easily extends to all higher dimensions.

Theorem For any dimension d there is a finite automaton such that a d-dimensional hypercube formed from n**d copies of this automaton will solve the connected 1s problem in O(n) time. □

This theorem, and its proof, can be applied to a variety of topological computations. We do not pursue this here, but do state a corollary linking PPAs and automata traversing finite dimensional tapes. The interested reader should consult [3,8].

Theorem For any dimension d and any predicate recognizable by a d-dimensional finite state automata, there is a d-dimensional PPA which recognizes it in linear time. □

4. FINAL REMARKS

The two-stage process of first designing an algorithm for a more powerful computer and then simulating this using clerks can be a useful method for solving a range of problems on a variety of models of parallel computation. This process tries to postpone as many gruesome details as possible, enabling one to get a reasonable start on a problem. Of course, the process also points out the desirability of working with the more powerful computer, with its ample wordsize, instead of shackling oneself with a model based on finite state automata.

Since clerks occupy several processors, one requirement for a clerk-based solution is that the number of powerful processors being simulated must be less than the total number of processors. In the circle problem this reduction was accomplished by using geometric considerations, while for the connected 1s problem it required a completely different method to start the small cubes. So far most image processing problems seem amenable to such a reduction.

We should mention that each clerk used herein is a linear automaton, but in general one could arrange them in other shapes. For example, squares or cubes may be useful for problems where each powerful processor does many computations. The reason for this is that a square clerk could keep a k-bit pseudo-register in a square $\sqrt{k}x\sqrt{k}$ and do all operations (addition, multiplication, division, square root, comparisons, etc.) in $O(\sqrt{k})$ time, instead of the O(k) needed if the pseudo-register is linear. The details of such

arithmetic will appear in Stout[13]. We do not yet know of an algorithm which requires these more rapid clerks, but we suspect that is because so few problems have been analyzed. Other possible variations on clerks include having pseudo-registers which are larger than a logarithmic function of the computer's size, having clerks whose size changes through the algorithm (perhaps starting with a fixed size and growing until their length is O(log(n))), or, for pyramid computers, having clerks which do not lie entirely in the base.

REFERENCES

1. C. Arcelli and S. Levialdi, Parallel shrinking in three dimensions, Comp. Graphics and Image Proc. 4 (1972), 21-30.

2. A. J. Atrubin, A one-dimensional real-time iterative multiplier, IEEE Trans. Elect. Comp. EC-14 (1965), 394-399.

3. W. T. Beyer, Recognition of topological invariants by iterative arrays, Ph. D. thesis, Mathematics, Massachusetts Institute of Technology, 1969.

4. C. R. Dyer, A fast parallel algorithm for the closest pair problem, Info. Proc. Lett. 11 (1980), 49-52.

5. M. Ercegovac, An on-line square rooting algorithm, Proc. 4th Symp. on Comp. Arith., IEEE Computer Society, 1978, 183-189.

6. D. Hirschberg, Parallel algorithms for the transitive closure and the connected components problems, Proc. ACM 8th Ann. Symp. Theory of Computing, 1976, 55-57.

7. D. E. Knuth, The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Addison-Wesley, New York, 1969.

8. S. R. Kosaraju, Fast parallel processing array algorithms for some graph problems, ACM Symp. on Theory of Comp. 11 (1979), 231-236.

9. S. Levialdi, On shrinking binary picture patterns, Comm. ACM (1972), 789-801.

10. D. Nassimi and S. Sahni, Finding connected components and connected ones on a mesh-connected parallel computer, SIAM J. Comp. 9 (1980).

11. B. Sakoda, Parallel construction of polygonal boundaries from given vertices on a raster, Penn. State. Univ. Comp. Sci. CS81-21.

12. Q. F. Stout, Drawing straight lines with a pyramid cellular automaton, to appear in Info. Proc. Letters.

13. Q. F. Stout, Arithmetic on finite dimensional automata, to appear.

14. S. L. Tanimoto and A. Klinger (eds.), Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures, Academic Press, New York, 1980.

15. K. S. Trivedi and M. D. Ercegovac, On-line algorithms for division and multiplication, IEEE Trans. Computers C-26 (1977), 681-687.