A SYSTOLIC ALGORITHM FOR THE FAST COMPUTATION OF THE CONNECTED COMPONENTS OF A GRAPH

Fernando J. Núñez and Mateo Valero

Departamento de Arquitectura de Computadores
Universidad Politécnica de Cataluña, c/ Pau Gargallo 5, 08028 Barcelona, Spain

ABSTRACT

This paper deals with the description of a systolic algorithm to solve the connected component problem. It is executed in a ring topology with N processors, requiring $O(N\log N)$ time without regard of the graph's sparsity. The algorithm partitioning issue is also addressed, indicating how to optimally map the computations into fixed-size rings or linear arrays. The proposed algorithm leads to simple processing elements, data addressing and control. These points make the systolic array highly implementable.

INTRODUCTION

Parallel graph algorithms is an active research area that has received so much attention in recent years. The architectural models considered in the conception of these algorithms may be grouped into: (1) processor arrays (SIMD), (2) multiple central processing unit computers (MIMD), (3) systolic arrays, and (4) associative processors [1].

It is said that a parallel model is reasonable if the number of processors each processor can communicate with is bounded by a constant. In systolic arrays only nearest neighbour communications are allowed. Mesh-connected computers (MCC) and systolic arrays are reasonable models. However, data routing requirements in MCC machines could be an obstacle for the development of fast parallel algorithms.

Systolic algorithms are good candidates for VLSI implementation because the simplicity of their processing elements (PEs) and their modest and regular inter-PE communication pattern. Systolic algorithms have the property of high implementability. For this reason they are good candidates for the design of special-purpose VLSI parallel machines.

Normally, when a new systolic algorithm is proposed, it has a number of (virtual) PEs that depends on one or more dimensions of the considered problem. For example, a square NxN PE array to multiply NxN matrices. They are problem size dependent algorithms. An interesting question arises when we try to map these algorithms into a fixed-size systolic array. In this situation, it is mandatory to decompose the problem (partitioning). Essentially, the original algorithm is split into subproblems directly solvable (executable) by the available fixed-size array. These subproblems must be properly and efficiently chained to attain the desired solution by combining partial results.

This work was supported by the Spanish Ministry of Education (CAICYT) under contract 0314-85.

Several partitioning methodologies have been developed. In [2] it is described an approach to achieve highly efficient partitionings for some basic Matrix Linear Algebra operations. Little work has been done in the partitioning of parallel graph algorithms. For example, in [3] a partitioning of the Transitive Closure suitable for the design of systolic arrays was presented.

A description of the contents of the rest of the paper follows. Firstly, we briefly review some outstanding parallel graph algorithms for the connected component problem, reporting their time complexity and computing model considered. After this, the solution strategy chosen for our systolic algorithm is described, focusing on the involved data structures. Then, the considered systolic array is described, as well as the execution details of the algorithm. The number of cycles required to solve the problem and other related parameters are indicated. Two types of computations mappings to fixed-sized arrays are commented. Finally, some conclusions are given.

PARALLEL CONNECTED COMPONENTS ALGORITHMS

Hereafter, we will consider an undirected graph G(V,E), where V is its vertex set, and E its edge set. The graph will have N vertices and M edges.

A good deal of work has been done for the parallel computation of the connected components (CC) of a graph [1]. Pioneering contributions to the vertex collapse algorithm cathegory were done by Hirschberg [4]. The components were obtained in $O(\log^2 N)$ steps using N^2 processors. The machine model considered was a SIMD-SM-R (SIMD with shared memory allowing any number of processors to access the same location simultaneously for reading but not for writing). Using the same model, Chin et al. [5] solved the problem with the same time complexity, but requiring only $N\lceil N/\log^2 N\rceil$ processors. Nassimi and Sahni [6] used the SIMD-MC (mesh-connected) model to find the CC in a square mesh of $\sqrt{N} \times \sqrt{N}$ processors in $O((2+d)\sqrt{N}\log N)$ time, being d the maximum vertex degree.Recently, Doshi and Varman [7] described an optimal parallel solution to the CC problem. It is solved in $O(N^2/p)$ time with a linear array of p computers (MIMD).

Our main interest is focused on the systolic array model. In [8] Savage used a linear systolic array with $N+1$ PEs to obtain the CCs in $O(N+M)$ steps. Hambrusch proposed a square $\sqrt{N} \times \sqrt{N}$ array to solve the problem in $O(N^{3/2})$ time [9]. By embedding tree structures in an N PE linear systolic array, Ashtaputre and Savage also needed $O(N+M)$ steps to solve the same problem very efficiently with an N PE linear array [10].

In this paper, we describe a systolic algorithm executed by an N PE ring that gives the components of a graph in $O(N\log N)$ steps, without regard of the graph's sparsity. Observe that algorithms with $O(N+M)$ complexity perform poorly for non-sparse graphs. In addition, it is a matrix-based algorithm because the graph is assumed to be represented by its adjacency matrix.

SOLUTION STRATEGY

In the following we assume that each vertex belonging to graph G is numbered with one integer in the range 1,..,N . For our purposes, the CC problem is solved when each vertex belonging to a certain component is labeled with the lowest numbered vertex in that component.

This paper is mainly devoted to adapt the solution strategy presented in [6] for a SIMD mesh-connected computer to a more restrictive systolic ring array. The strategy belongs to the vertex collapse approach. This approach is summarized in the next paragraph.

The graph's components are obtained through several iterations. At the beginning of each iteration, the partial components obtained up to that point are combined (collapsed) into greater ones, reducing its number by a factor of, at least, two. Initially, each vertex is a component by itself. Therefore, in the worst case, $\lceil \log N \rceil$ iterations are required.

In the solution process several auxiliary data structures appear. A min-tree (MT) is a directed tree where each vertex points to another vertex with lesser number. The height of a tree is defined as the maximum distance between every leaf and the tree root, which points to itself, and whose height is zero. A reduced min-tree (RMT) is a MT of height at most one. A tree-loop (TL) is formed when the root of a directed tree points to one of its descendants. Figure 1 shows examples of these structures; (sub)graphs (b), (c), and (d), are TLs, MTs, and RMTs respectively.

Each connected component is represented by a RMT. The lowest numbered vertex in a RMT is the root. Every vertex in a RMT points to the root (including itself), which gives the component number.

The number of RMTs is reduced in each iteration. The systolic ring presented in this paper requires four steps to complete each iteration:

Step 1: Find the minimum RMT adjacent to each vertex.
Step 2: Make the root of each RMT point to the root of the minimum RMT adjacent to its vertices.
Step 3: Convert the resulting TLs into a MT forest.
Step 4: Convert the MT forest into a RMT forest.

At this point, things are ready for the next iteration.

These four steps can be stated more formally by the following expressions:

Step 1: $m(v) := \min(c(u))$ such as $c(u) \neq c(v)$ and $a(v,u) = 1$; ∞ otherwise
Step 2: $c(v) := \min(m(u))$ such as $c(u) = v$; if ∞ then $c(v)$
Step 3: $c(v) := c(c(v))$ if $c(v) > v$; $c(v)$ otherwise
Step 4: $c(v) := \min(c^k(v))$

where v and u are vertices. At the beginning of each iteration c(v) is the component number vertex v belongs to. The elements of the adjacency matrix are denoted as a(v,u) for $1 \leq v,u \leq N$, and "min" denotes the minimum function.

After the first step, m(v) points to the root of the lowest numbered component adjacent to vertex v. If vertex v is surrounded by vertices in the same component c(v), m(v) is assigned the value ∞.

In the second step, root vertices point to the root of the minimum neighbouring component, if it exists, by consulting to the vertices they are pointed by.

After this step, there may be roots r such as $c(r) > r$. This could lead to trees with cycles (in fact TLs). It is mandatory to obtain MTs in order to allow the systolic array to reduce them to RMTs, representing the new components. Fortunately, this is possible because $c(c(r)) \leq r$ [1]. Once performed this operation over the proper roots, we will have only MTs.

The new component of each vertex will be its lowest numbered ancestor, i. e., the root of the MT it belongs to. This reduction to RMTs is performed in the fourth step.

Figure 1 illustrates this process for the nine-vertex graph depicted in (a). In figure 1 (b) the result after step 2 is presented. The directed edge going out from each vertex v represents c(v). Note that in the first iteration the resulting (directed) graph after steps 1 and 2 is the same because all vertices are also roots. Figure 1 (c) shows the MT forest obtained after computing c(c(v)) for vertices 1, 2, and 4. Finally, in figure 1 (d) the RMT forest representing the new components can be observed. For this graph, a second iteration is enough to obtain a single component or RMT rooted at vertex 1.

Table 1 could help to clarify this process. There we can see m(v) after step 1, and c(v) after steps 2, 3, and 4 for the two iterations required by the example graph.

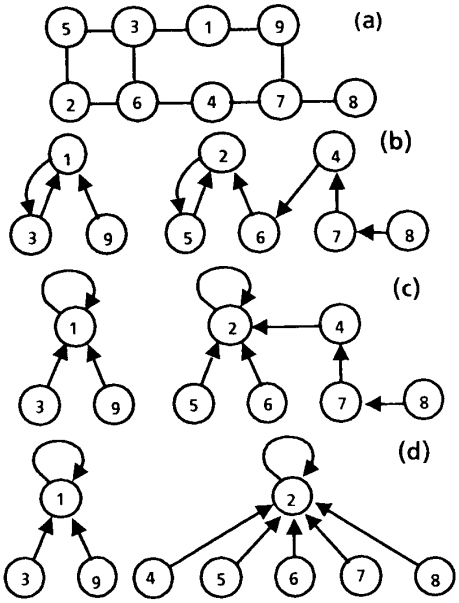

Figure 1. (a) An example graph and (b) the TLs, (c) MTs, and (d) RMTs after the first iteration.

	Step		vertices 1	2	3	4	5	6	7	8	9
Iterarion 1	1	m(v)	3	5	1	6	2	2	4	7	1
	2	c(v)	3	5	1	6	2	2	4	7	1
	3	c(v)	1	2	1	2	2	2	4	7	1
	4	c(v)	1	2	1	2	2	2	2	2	1
Iterarion 2	1	m(v)	∞	∞	2	∞	1	1	1	∞	2
	2	c(v)	2	1	1	2	2	2	2	2	1
	3	c(v)	1	1	1	2	2	2	2	2	1
	4	c(v)	1	1	1	1	1	1	1	1	1

Table 1

THE SYSTOLIC RING ARRAY

A ring topology has been selected because the four steps fit quite naturally in it. The systolic ring is depicted in figure 2. It has N processing elements (PEs), denoted as PE(1),...,PE(N). As it is shown, the inter-PE links are unidirectional. In the figure it is also shown the external memory where the adjacency matrix is held. PE(v) can read row (or column) v. In addition, the input and output ports, and the registers present in a PE are also detailed. Registers I and A hold the values read from their corresponding input ports.

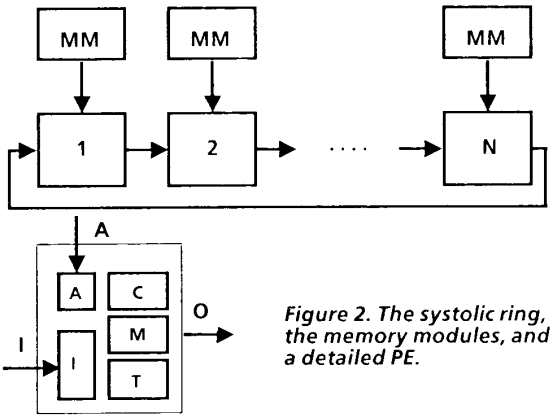

Figure 2. The systolic ring, the memory modules, and a detailed PE.

PE(v) is associated to vertex v. The main functions of registers C and M are, respectively, to hold the component number and minimum adjacent RMT to the vertex associated to that PE. Register T is a time or cycle counter. At the beginning of each cycle, registers I latch the values present at their I ports and register T is incremented.

The PEs perform a few simple operations; one operation is executed per cycle. They are completely specified by indicating the contents of the registers, and the value assigned to the output port O at the end of the cycle. All the PEs perform the same operation at the same time. A single control unit suffices to generate the operation codes.

Memory addressing is very simple. The adjacency matrix A is read only in step 1. PE(v) first reads element a(v,v) which is a dummy value, and then the remaining row elements in descending circular order. Consequently, address generators reduce to module-N down counters that must be properly initialized.

EXECUTION

The program executed by each PE follows.

```
do ⌈log(N)⌉ times

 /* Step 1 */
 M:=∞, O:=C;
 do N-1 times
  O:=I, if A=1 and C≠I then M:=min(M,I);
 end do

 /* Step 2 */
 O:=M, T:=1;
 do N-1 times
  if T=v-C+1 then O:=min(M,I)
             else O:=I;
 end do
 if C=v and I≠∞ then C:=I;

 /* Step 3 */
 O:=C, T:=1;
 do N-1 times
  O:=I, if T=N-C+v+1 then C:=I;
 end do

 /* Step 4 */
 O:=C, T:=1;
 do N-1 times
  O:=I, if T=v-C+1 then C:=I;
 end do

end do
```

Let us explain each step.

Step 1: The component number of each PE (vertex) visits the N-1 remaining PEs traveling through the ring. By consulting the read bits in A, each PE knows whether a given component is adjacent or it is not. By performing "min" operations, the minimum adjacent RMT is found and left in register M. When a given vertex and all its adjacents belong to the same component, its register M will store the value ∞.

Step 2: Now, each vertex sends its minimum adjacent RMT (contents of M). When this number reaches a vertex that points to the one that sent it (its root), it is actualized with the minimum between itself and the minimum RMT adjacent to the visited vertex. Note that the cycle counter register T is used to detect, for a vertex v, when the value sent by vertex c(v), and maybe updated by other vertices, is present in register I.

Step 3: In this moment, a root r could point to another root s such as $c(r)=s>r$. Hence, we do not have MTs. Fortunately, s must point to other root t verifying $t \leq r$ [6]. Using register T, PE(r) detects when t reaches register I, storing it in C. Once all the concerned PEs have completed this operation, a MT forest results.

Step 4: In a MT, there is a descending path from every vertex to the root. Now, each vertex v sends to the ring the number of its father c(v), held in C. A given vertex is visited by the fathers of the other vertices in descending order. Again, register T is used to detect when the values sent by the ancestors of a given vertex are present in I. This reduction step produces a RMT forest.

ALGORITHM TIME, COMPLEXITY, SPEEDUP, AND COST

Steps 1, 3, and 4, all require N cycles, while step 2 spends N + 1 cycles. Then, 4N + 1 cycles are needed per iteration. This gives a gobal execution time of $4(N+1)\lceil \log N \rceil$ cycles. Consequently, the proposed algorithm belongs to the $O(N\log N)$ time complexity class.

The optimal sequential CC algorithm based on the graph's adjacency matrix representation, belongs to the $O(N^2)$ class. This can be easily seen if we observe that each element of the matrix has to be read al least once.

Assume we define the speedup of a parallel algorithm as the ratio of the time spent by the fastest sequential algorithm for that problem to the time needed by the parallel algorithm. The speedup achieved is $O(N/\log N)$, almost linear for large values of N.

Defining the cost of an algorithm as the time spent times the number of processors used, this systolic CC algorithm has $O(N^2\log N)$ cost. It is $O(\log N)$ times more costly than the optimal sequential algorithm.

A WORD ABOUT PARTITIONING AND IMPLEMENTATION

This section deals with the mapping of the previously described systolic algorithm to a fixed-size array with p processors. Assume, without loss of generality, that $p = kN$, where k is an integer.

Hereafter, we will consider the case where partitioning is achieved by coallescing or grouping the initial N PEs into p sets of N/p PEs each.

The ring topology is preserved if consecutive PEs are grouped. Nevertheless, although rings do not present excessive implementation problems, it could be more convenient to map the computations into a linear array. Figure 3 illustrates this point.

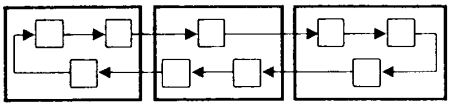

Figure 3. Coallescing PEs when N = 9 p = 3.

Control details are obvious, but it is worth to mention that some optimizations due to grouping virtual PEs into one physical PE are now possible. An example could be the sharing of register T by the PEs mapped to the same physical processor. Note that the resulting PEs have four links (with the exception of the top left and top right PEs).

This p-processor array is able to obtain the components in $O((N^2/p)\log N)$ time.

The resulting partitioned implementation observes three principles [11]:

(1) Minimization of the external hardware and communication overhead.

(2) Minimization of the additional delays.

(3) Minimization of the control and PE complexity overhead.

Let us make a last comment about speed. Recently, a smart optimal CC algorithm has been presented [7]. It requires $O(N^2/p)$, $1 \leq p \leq N$ with a p-processor linear MIMD array.

It would be interesting to compare the absolute execution times of that algorithm and the one described here, for different values of N, when both algorithms are mapped to the same MIMD computer. Although one algorithm is asympthotically optimal, the other has simpler operations.

CONCLUSIONS

Finding the connected components of a graph is an important problem in graph theory. Several parallel algorithms to solve this problem have been proposed. They were conceived to be executed in abstract models of parallel machines, as well as for systolic arrays.

Systolic arrays are interesting models because they allow to achieve a high degree of parallelism, and they are very implementable. Systolic arrays are well suited for VLSI-based machines.

We propose a systolic algorithm to solve the CC problem in $(4N+1)\lceil \log N \rceil$ steps without regard of the graph's sparsity. This parallel algorithm has $O(\log N)$ times more cost than the optimal sequential algorithm. It achieves almost linear speedup for large values of N.

The partitioning issue is addressed by indicating how to map the algorithm into a smaller p-processor array with ring or linear topology. The time to solve the partitioned problem is $O((N^2/p)\log N)$.

REFERENCES

[1] M.J. Quinn, N. Deo, Sept. 1984. "Parallel Graph Algorithms", Computing Surveys 16, 3.

[2] J.J. Navarro, J.M. Llabería, M. Valero, July 1987. "Partitioning: An Essential Step in Mapping Algorithms into Systolic Array Processors", IEEE Computer Magazine, vol. 20, no. 7.

[3] F.J. Núñez, N. Torralba, Aug. 1987. "Transitive Closure Partitioning and Its Mapping to a Systolic Array", Proc. of the 16th Int. Conf. on Parallel Processing.

[4] D.S. Hirschberg, May 1976. "Parallel Algorithms for the Transitive Closure and the Connected Component Problems", Proc. of the 8th Annual ACM Symp. on the Theory of Computing.

[5] F.Y. Chin, J. Lam, I.N. Chen, Aug. 1981. "Optimal Parallel Algorithms for the Connected Component Problem", Proc. of the 10th Int. Conf. on Parallel Processing.

[6] D. Nassimi, S. Sahni, Nov. 1980. "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer", SIAM J. on Computing 9, 4.

[7] K.A. Doshi, P.J. Varman, April 1987. "Optimal Graph Algorithms on a Fixed-Size Linear Array", IEEE Trans. on Computers, vol. C-36, no. 4.

[8] C.D. Savage, Jan. 1984. "A Systolic Design for Connectivity Problems", IEEE Trans. on Computers, vol C-33, no. 1.

[9] S.E. Hambrusch, May 1983. "VLSI Algorithms for the Connected Component Problem", SIAM J. on Computing, vol. 12, no. 2.

[10] S. Ashtaputre, C.D. Savage, May 1985. "Systolic Arrays with Embedded Tree Structures for Connectivity Problems", IEEE Trans. on Computers, vol C-34, no. 5.

[11] S.Y. Kung, 1987. "VLSI Array Processors", in SYSTOLIC ARRAYS, W. Moore, A. McCabe, R. Urquhart, eds., Adam-Hilger.