Abstract — The applicability of the systolic/wavefront array concept for a further class of algorithms is evaluated. A representative of this class is Ford's algorithm for the solution of the single-source-shortest-path problem under consideration of sparsity properties of the graph. Data transportation turns out to be the crucial point of the systolic realization. In the case of Ford's algorithm two different kinds of data transport problems arise which are solved by broadcasting within a connected region and by use of a two-dimensional systolic sorting procedure. Further, a comparison between the systolic version of Ford's algorithm and systolic realizations of Floyd's method, which is the corresponding direct algorithm, are given.

Introduction

Due to the increasing demand for high speed computation in many application fields the area of parallel computing is gaining more and more importance. There are quite a lot of different concepts for the realization of a parallel computer. Within the last years the idea of parallel processing on systolic/wavefront arrays has attracted much attention as the properties of such arrays, as e.g. regularity, modularity and locality, make them well suited for VLSI implementation [2]. Another advantage is the lack of memory conflicts as each processor only uses its own local memory. Many important algorithms, as e.g. matrix multiplication, LU-decomposition, transitive closure, can optimally be executed on systolic arrays after being transformed into the form of a regular iterative algorithm [6]. But in the case of sequential computing often methods are preferred, which do not allow a representation as a regular iterative algorithm. Hence, the question arises, whether it is possible to efficiently execute these methods on a systolic array as well.

This paper deals with this kind of problems in the case of a parallel solution of the single-source-shortest-path problem in a sparse graph. Shortest path algorithms belong to the most important algorithms of combinatorial optimization and play an important role in the field of computer-aided design. There, they can either be applied directly, as e.g. in the case of compaction and routing or they serve as important subroutines in more complex algorithms, as e.g. transportation algorithms for retiming and assignment algorithms for placement. Further, a lot of systolic array processors for shortest path computations have been proposed, e.g. [3,7]. As the main operations are addition and comparison the area requirements of an elementary processing cell are small in comparison to linear algebra processors and a large number of processor cells can be implemented on an integrated circuit or even on a wafer. But, these systolic realizations are all based on Floyd's procedure [4], which is quite inefficient for many kind of problems as shown in the next section. Here, Ford's algorithm [4] is often better suited than Floyd's method. But the only known parallel realizations of Ford's algorithm make excessive use of data broadcasting and global communication, see e.g. [5], and therefore do not fit the properties of systolic arrays. This paper presents a systolic realization of Ford's method and gives a comparison between various methods for calculation of shortest paths.

Shortest Path Algorithms

This paper is mainly concerned with the solution of a single-source-shortest-path problem on a graph with negative and positive edge weights. This solution can be described by the means of a directed tree. Here, we define the depth $t$ of the tree to be the maximal number of edges of any directed path within the tree. The length of a shortest path starting at the source vertex and ending a vertex $v$ is denoted by $u(v)$ and $D = [d(i,j)]$ is the distance matrix of the graph. Now, Ford's algorithm can be described as follows, where we use the notations of the corresponding path algebra (fig.1):

### Fig.1 Ford's algorithm

<table>
<thead>
<tr>
<th>Presumption $d(i,i)=0$</th>
<th>shortest path path algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_k^{(1)} = [0,+,\ldots,+]$</td>
<td>$u_k^{(n)} = [1,0,\ldots,0]$, $k = 0$</td>
</tr>
<tr>
<td>$u_{k+1}^{n+1} = u_k^n + D$</td>
<td>$u_k^n \cdot D$, $k = n+1$</td>
</tr>
<tr>
<td>Repeat 2. until $u_k^{n+1} = u_k^n$ or $k = n$</td>
<td></td>
</tr>
</tbody>
</table>

Here, $n$ is the number of vertices and $e$ is the number of edges in the graph. In the case of sequential computing, Ford's algorithm in this form requires $t+e$ operations. Floyd's procedure on the
other hand solves the more general all-pairs-
shortest-path problem and always requires \( n^3 \)
operations. If the graph is sparse, i.e., \( e \ll n^2 \),
and \( t \ll n \), it is obviously quite inefficient to
solve single-source-shortest-path problems by use
of Floyd's method due to memory and time require-
ments. At first sight the situation is almost the
same in the case of parallel computing: Floyd's
method requires \( O(n) \) time complexity with \( n^2 \)
processors and Ford's algorithm needs a time com-
plexity of \( O(t \log n) \) and \( e \) processors [5]. But while
the parallel algorithm of Ford makes excessive use
of global memory and data broadcasting, it is pos-
sible to execute Floyd's method on a systolic ar-
ray [3,7]. Therefore, it is the objective of the
further sections of this paper to evaluate ways of
a systolic execution of Ford's algorithm.

Parallel Execution of Ford's Algorithm

Ford's algorithm obviously consists of a kind of
repeated matrix-vector multiplication. The ele-
mentary block of this operation is the calculation of
an inner product. By using a tree structure (fig.2)
an inner product can be computed with a time com-
plexity of \( O(\log n) \), if \( n \) processors are available.
As the \( n \) inner products of a matrix-vector multi-
plication do not depend on each other, it is poss-
able to receive the same time complexity for the
determination of the whole resulting vector \( u_{k+1} \) by
using \( n \) tree structures in parallel.

\[
\begin{align*}
  u_k(i) + & \frac{d(i,j)}{v} \\
\end{align*}
\]

**Fig. 2** Inner product computation in a tree

If the graph has a fixed sparsity structure, this
properties can easily be taken into account in
order to reduce the number of processor elements.
As each iteration is based upon the result of
the previous one, however, a parallel execution of
different iterations is not possible. These facts
yield a time complexity of \( O(t \log n) \) for the pa-
allel execution of Ford's algorithm by using \( e \)
processor cells.

In order to receive a systolic realization of
Ford's algorithm, which should be superior to
Floyd's method in the case of a sparse graph, we
demand the compliance of following conditions:

1. (C1) Number of processor cells = number of edges
   within the graph
2. (C2) Applicability of the architecture for all
   problems with \( n \leq n_{\text{max}} \) and \( e \leq e_{\text{max}} \).
3. (C3) Local processor interconnection structure.

(C1) and (C2) imply a variable interconnection
structure as the assignment of the processor cells
to the inner product trees depends on the sparsity
structure of the graph. Further, (C1) prevents the
use of a tree interconnection structure. This con-
tradiction can be solved by introducing additional
transport cells into the tree structure. If we do
so, there will be an increase of time complexity.
Therefore, the operation scheme of a tree can be
abandoned. Obviously, the major problem of a systo-
ic execution of Ford's algorithm under considera-
tion of a sparsity structure is the solution of
data transport problems, which is the subject of
the next section.

Data Transportation

A parallel solution of a matrix-vector mul-
tiplication requires the solution of two data transport
problems:

1. (T1) Transport of a data \( u_k(i) \) to all elements
d(i,j) of row i of the distance matrix:
The broadcasting problem.
2. (T2) Transport of all data \( u_k(i) \) \( d(i,j) \) related
to column j of the distance matrix to a
position where \( u_k(i) \) is formed by
minimization: The funnel problem.

A systolic realization of an algorithm is always
connected with an arrangement of data on an array.
Although the actual arrangement may change during
the run of the algorithm, usually the structure of
the arrangement remains invariant. In most matrix
operations, for instance, the data elements are
arranged according to their matrix indices, while
there may be an exchange of complete rows and
complete columns, e.g. [8]. If we define a processor
array to be a graph, where the cells are vertices
and the interconnections are edges then a well sui-
ted arrangement of data fulfills subsequent condi-
tion:

- (S1) There is exactly one path between the source
  vertex of a data and each destination vertex
  of the data, where all intermediate vertices
  are destination vertices, too.

Sometimes, it is accepted that some intermediate
vertices only serve for data transport without
executing any operations, but in this case the effi-
ciency of the array is reduced.

In general, it is not possible to arrange all
elements of an \( nxn \) matrix in a linear order \( p \) such that

\[
|p(d(1,1)) - p(d(1,k))| \leq O(\sqrt{n}) \quad \text{and} \quad |p(d(1,1)) - p(d(k,1))| \leq O(n) \quad \text{for all } i, j, k.
\]

Therefore, an efficient realization of Ford's
algorithm requires an array of dimension 2 at
least. Further, it can be seen easily that there is
a contradiction between the conditions (S1) and
(C1) in the general case. Therefore, a modification
of (S1) is introduced:

- (S2) There is at least one path as described in
  the condition (S1).

423
If (S2) is fulfilled for one data then source and all destination vertices of the data form a connected region (fig. 3).

The possibility of several data paths with different lengths according to (S2) may complicate the transport problem. Here, it is easy to handle Ford's algorithm due to following property of the minimum operation, which is not valid for the addition, i.e. for some other algorithms belonging to the same path algebra:

\[(M) \quad \text{Min}(a,b) = \text{Min}(a,\text{Min}(a,b))\]

But unfortunately, it is not possible as well to comply (S2) and (Cl) for all problems if a 2-d mesh connected array is used. With other words, we cannot always find an arrangement of all edges of the problem graph on a 2-d mesh connected array, where each column and each row form a connected region. This statement can be verified easily if we look for such an arrangement for a n+1x(n+1) matrix on a nxa matrix after scratching one element of each row and each column. Now, two approaches are possible. Either the introduction of a small number of additional edges in the problem graph with \(d(i,j) = +\infty\) is accepted or the number of processor interconnections is increased. There are quite a lot of difficulties connected with both ideas. But the main drawback is the necessity of preprocessing which prevents an efficient systolic realization. Otherwise, a more complex data transport problem must be solved, which is the matter of the next section.

**Systolic Solution**

Now, it must be assumed that the elements of a row or a column of the distance matrix are distributed within the array. The data transport problem can principally be solved by use of a sorting algorithm [1] if there is an appropriate order of the data elements. Here, the use of a lexicographical order (fig. 4) is advantageous.

The wrap-around connections of figure 4 do not mean any violation of the locality property of a systolic array as they can easily be transformed into local connections by folding the array around the vertical middle axis. Beside enabling the execution of a sorting algorithm it creates a connected region for each row of the distance matrix and does not require complex permutation operations before the start of the algorithm. Therefore, the broadcasting problem (T1) can be solved in a simple way by exploiting property (M).

In order to choose a well suited sorting algorithm the structure of the processor array must be considered. Conceptually, the array is divided into two planes (fig. 5).

Each elementary cell of the operation plane (OP) contains an edge weight \(d(i,j)\) and a level \(u(i)\). It can execute following operations:

- \(u(i)+d(i,j)\) output (TP)
- \(u(i)\) output (OP)
- \(\text{Min}(u(i),u(i)\text{ input (TP/OP)}\)

The cells of the transport plane perform the transport of a new level \(u(j)\) to any operation cell containing \(d(j,*).\) In order to avoid a transport without destination, it must be demanded that there is no vertex in the graph with the out degree 0. Otherwise an additional loop \((j,j)\) with \(d(j,j)=0\) can be introduced.

Consequently some requirements for the sorting algorithm arise:

- two dimensional systolic sorting,
- low time complexity of about \(O(\log n)\),
- well suited for VLSI implementation.

Further, if there is an appropriate criterion then a priority transport should be supported in order to compensate different distances between source and destination cell of levels. Beside the algorithm must manage the permanent disturbance of order caused by the creation of levels \(u(j)\). A comparison between various sorting procedures, see [10], shows that a new short-periodic algorithm [9] is best suited for this task. Only its time complexity is slightly increased by the factor \(\log n\) compared to the optimal value of \(O(\log n)\).
At last, it remains to examine the coupling of the planes. While the transition from (TP) to (OP) is done by minimization, the opposite transition may cause problems. Here, it is a precondition that the corresponding transport cell does not contain a level. Although our computer simulations do not show any influence of time complexity due to the comparatively small number of levels within the transport plane, this effect may lead to a high worst case complexity. But this can easily be avoided by executing a reverse transport of a dummy after each completed transport of a level. This must be paid with an increase of the time complexity by the factor 2 in the average case.

Comparison and Concluding Remarks

A comparison between the properties of a systolic execution of Floyd's method on one- and two-dimensional arrays and the systolic version of Ford's algorithm is given in table 1.

<table>
<thead>
<tr>
<th></th>
<th>Floyd</th>
<th>Ford</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-d</td>
<td>2-d</td>
</tr>
<tr>
<td>time</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>processor</td>
<td>n</td>
<td>$n^2$</td>
</tr>
<tr>
<td>area</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>AT</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

Tab.1 Properties of systolic realizations

The superiority of Ford's algorithm in the case of a sparse graph can be seen from table 2.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Floyd</th>
<th>Ford</th>
</tr>
</thead>
<tbody>
<tr>
<td>e t</td>
<td>1-d</td>
<td>2-d</td>
</tr>
<tr>
<td>$n^2$ n</td>
<td>$n^2$</td>
<td>$n^2+log n$</td>
</tr>
<tr>
<td>n $\sqrt{n}$</td>
<td>$n^2$</td>
<td>$n+log n$</td>
</tr>
<tr>
<td>n log n</td>
<td>$n^2$</td>
<td>$n+log n$</td>
</tr>
</tbody>
</table>

Tab.2 Time complexity for different problems

The second problem arises, for instance, if a shortest path algorithm is used for an one-dimensional layout compaction. The algorithm can be accelerated by using priority transport according to the generation number of a level $u(i)$.

References