The parallel computation of minimum cost paths in graphs by stream contraction

V. Pan *

Department of Mathematics and Computer Science, Lehman College, CUNY, Bronx, NY 10468, USA and Computer Science Department, SUNY Albany, Albany, NY 12222, USA

J. Reif **

Computer Science Department, Duke University, Durham, NC 27706, USA

Abstract

We accelerate by a factor of $\log n$ and with no increase of the processor bound our previous parallel algorithm for path algebra computation in the case of the minimum cost path computation in an n-vertex graph and, more generally, wherever the path algebra has an order relation defined by its $\oplus$ operation. The acceleration is obtained by means of a novel technique of stream contraction.

Keywords: Parallel algorithms, computational complexity, path algebras

1. Introduction. General stream contraction technique and the main result for path algebra

It has been well recognized that numerous important computations of paths in an n-vertex graph $G=(V, E)$ can be both unified and simplified by means of reducing them to the evaluation (over a fixed dioid, also called semiring or path algebra) of the quasi-inverse A^* given an $n \times n$ matrix $A = A(G)$ associated with the graph G (see [1,2,4,6], for details).

In the present paper we propose a novel general technique of *stream contraction* for the acceleration of parallel algorithms by means of their systolic rearrangement and demonstrate its power by accelerating the recent parallel algorithms of [6] for several important path algebra computations. To show how this technique of stream contraction works, we will now describe two algorithms. The first generalizes the algorithm of [6] and the second represents its new acceleration. We assume that A, X and W (with subscripts) are matrices related to each other through the functions f, g, p and q; d and $K(h)$ for $h=0, 1, \ldots, d$ are fixed positive input integers, $A_{0,0}$ is an input matrix and $A_{h,K(h)} = W_{h,K(h)}$, for $h=0, 1, \ldots, d$, are the out-

* Supported by NSF Grants DCR 85-07573, CCR8805782 and CCR 9020690 and by PSC CUNY Awards No. 668541, No. 669290 and No. 661340.

** Supported by Air Force Contract AFOSR-87-0386, ONR Contract N00014-87-K-0310, DARPA/ISTO Contract N00014-88-K-0458, ARO Contract DAAL03-88-K-0195, and NASA/CESDIS Subcontract 550-63 NAS 5-30428 URSA.

Preprint of paper appearing in *Information Processing Letters*, Vol. 40, October 25,1991, pp. 79-83.

put matrices. For path algebra computations, the matrices represent the input, output and auxiliary graphs, and the output defines the paths and their properties.

Algorithm 1.

for $h = 0, 1, \ldots, d$ **do**
  for $k = 0, 1, \ldots, K(h) - 1$ **do**
    $X_{h,0} = p(A_{h,0})$
    $W_{h,0} = f(X_{h,0})$
    $W_{h,k+1} = q(W_{h,k})$
    $A_{h+1,0} = g_h(W_{h,K(h)}, A_{h,0})$
    $A_{h,K(h)} = A_{h,0}$
  end for
end for

Algorithm 2.

for $h = 0, \ldots, d$ **do**
  $X_{h,0} = p(A_{h,0})$
  $W_{h,0} = f(X_{h,0})$
  $A_{h+1,0} = g_h(W_{h,0}, A_{h,0})$.
end for
for $h = 0, \ldots, d$ **do**
  for $k = 0, \ldots, K(h) - 1$ **do**
    $W_{h,k+1} = q(X_{h,k}, W_{h,k})$
    $A_{h+1,k+1} = g_h(W_{h,k+1}, A_{h,k})$
    $X_{h,k+1} = p(A_{h,k+1})$
  end for
end for

Applying Algorithms 1 and 2, for appropriate functions f, g, p, q and $K(h)$, to certain path algebra computations, we will arrive at the same desired output (see below). If we assume that each evaluation of f, g, p or q can be performed in a single time-step on a single processor, then Algorithm 1 takes $\sum_{h=0}^{d-1} K(h)$ time-steps on a single processor and cannot run faster even if more processors are available, whereas Algorithm 2 takes $3(d + 1 + \max_h K(h))$ steps using $d + 1$ processors because the second "for" loop (for $h = 0, \ldots, d$) can be performed concurrently.

In the particular case of the algorithm of [6] as Algorithm 1, we achieve a speedup of $c \log n$ for a positive constant c; our acceleration technique requires no increase of the asymptotic processor bounds and applies to a large and important class of the minimum cost path computations in the planar graphs or planar digraphs G having no negative cost cycles. Furthermore, we may allow negative cost cycles in G provided that we compute paths consisting of at most $n = |V|$ edges over dioids (semirings) S whose operation $+$ defines an order relation, so that $a + b \leqslant a$, for any pair of the elements of S [1,2,4,6]. Moreover, similarly to [6], we may relax the assumptions that the input graph or digraph G is planar; what we actually need is just an $s(n)$-separator family defined by a fixed separator tree of G, where $s(n) = O(n)$; in particular, for planar graphs, $s(n) = O(\sqrt{n})$.

With appropriate modifications, the stream contraction technique can be applied to many other computational problems, in particular, see [3].

Hereafter, for simplicity we assume that G is an undirected graph (the extension to digraphs G is straightforward, see [6]).

2. The "old" algorithm

We first recall that the algorithms of [6] compute the matrix A^* by means of computing its recursive factorization (given by equations (8) and (9) of [6]),

$$A_0 = PAP^{\mathrm{T}}, \qquad A_h = \begin{bmatrix} X_h & Y_h^{\mathrm{T}} \\ Y_h & Z_h \end{bmatrix}, \qquad (*)$$

$$A_{h+1} = Z_h \oplus Y_h X_h^* Y_h^{\mathrm{T}},$$

$$A_h^* = \begin{bmatrix} I & X_h^* Y_h^{\mathrm{T}} \\ 0 & I \end{bmatrix} \begin{bmatrix} X_h^* & 0 \\ 0 & A_{h+1}^* \end{bmatrix} \begin{bmatrix} I & 0 \\ Y_h X_h^* & I \end{bmatrix}, \qquad (**)$$

where P denotes a permutation matrix, $h = 0, 1, \ldots, d$, $d = O(\log n)$, $n = |V|$, the number of vertices of the input graphs $G = (V, E)$, and where the choice of P and the partition of the $n_h \times n_h$ matrix A_h into its blocks X_h, Y_h, Z_h and Y_h^{T} depends on the separator structure of the graph and is computed in the nested dissection algorithm of [5], which is the basis of the algorithm of [6]. The algorithm of [6] consists in recursive

evaluation of X_h^* and A_{h+1}, for $h=0, 1, \ldots, d$, where in the case of the minimum cost paths with no negative cost cycles the quasi-inverse X_h^* (given X_h) is computed by using the matrix equations $X_h^* = (I \oplus X_h)^i$ for all $i \geqslant s(n_h) - 1$ (see [1–3,6]), and then the matrices A_{h+1} (and thus X_{h+1}) are computed by using (*). The computation is effective because n_h decreases as a geometric progression as h grows and X_h is a block diagonal matrix with diagonal blocks of smaller size (of the size of the separators), as these are well-known properties of the generalized nested dissection algorithms, to which class the algorithm of [6] belongs. On the other hand, the evaluation of X_{h+1}^* only starts when A_{h+1} and thus its submatrix X_{h+1} has been computed, and therefore, only when the quasi-inverse X_h^* has been evaluated (see (*)). Therefore, the parallel time bound of the algorithms of [6] equals the sum of the respective bounds for computing X_h^* for all h, $h=0, 1, \ldots, d$, d being of the order of $\log n$. In the present paper, we start computing X_{h+1}^* before we end computing X_h^* and this way speed up the parallel computations.

To reconcile this algorithm with Algorithm 1 of the Introduction, we set

$$A_{0,0} = A_0, \qquad X_{0,0} = X_0,$$

$$p(A_h) = X_h \quad \text{(according to (*))},$$

$$f(X) = I \oplus X, \qquad q(W) = W^2,$$

$$g(W, A_h) = Z_h \oplus Y_h W Y_h^{\mathrm{T}} \quad \text{(according to (*))}.$$

Then we observe that $W_{h,k+1} = (I \oplus X_h)^{2^k}$ and, therefore, $W_{h,K(h)} = X_h^*$ since $2^{K(h)} \geqslant s(n_h) - 1$, $A_{h,K(h)} = A_{h,0} = A_h$ for all h (as desired).

3. The new algorithm

Our new improvement relies on a systolic rearrangement of the algorithm of [6], so that computing X_{h+1}^* (and therefore, A_{h+1}^*) starts before the matrix X_h^* has been evaluated.

In our new accelerated factorization, we define a sequence of matrices $A_{h,k}$; we proceed recursively in h, for $h=0, 1, \ldots, d$, and in k, for $k=0, 1, \ldots, d + \lceil \log_2 n \rceil$, starting with $A_{0,0} = A_0$. For all h and k, we let

$$A_{h,k} = \begin{bmatrix} X_{h,k} & Y_{h,k}^{\mathrm{T}} \\ Y_{h,k} & Z_{h,k} \end{bmatrix}, \tag{1}$$

$$W_{h,0} = I \oplus X_{h,0},$$
$$A_{h+1,0} = Z_{h,0} \oplus Y_{h,0} W_{h,0} Y_{h,0}^{\mathrm{T}}, \tag{2}$$

$$W_{h,k+1} = (X_{h,k} \oplus W_{h,k})^2, \tag{3}$$

$$A_{h+1,k+1} = Z_{h,k} \oplus Y_{h,k} W_{h,k+1} Y_{h,k}^{\mathrm{T}}. \tag{4}$$

We first compute the matrices $W_{h,0}$, $A_{h+1,0}$, for all h, by using (2); then we recursively apply (3) and (4) in order to compute first the matrices $W_{h,1}$, $A_{h+1,1}$, for all h, then the matrices $W_{h,2}$, $a_{h+1,2}$, for all h, and so on, ending with $W_{h,k}$ and $A_{h+1,k}$ for $k = d + \lceil \log_2 n \rceil$. Finally, we set $X_h^* = W_{h,k}$, $A_h = A_{h,k}$, for $k = h + \lceil \log_2 n \rceil$ and for all h, which defines the desired recursive factorization (*), (* *).

In an alternate version of this algorithm, we end the computation with $k = \lceil \log_2 (s(n_0) - 1) \rceil$ and arrive at (*), (* *) by setting $X_h^* = W_{h,k}$, $A_h = A_{h,k}$ for $k = \lceil \log_2 (s(n_h) - 1) \rceil$ and for all h (see Remark 2 below). (We give two versions of the algorithm and of the correctness proof to demonstrate more fully some variations of the stream contraction technique.)

To reconcile the latter algorithm (in its second version) with Algorithm 2 of the Introduction we set

$$X_{h,k} = p(A_{h,k}) \quad \text{(according to (1))},$$

$$f(X) = I \oplus X, \qquad q(X, W) = (X \oplus W)^2,$$

$$g_h(W, A_{h,k}) = Z_{h,k} \oplus Y_{h,k} W Y_{h,k}^{\mathrm{T}}$$

according to (1) and (4)).

In the next section (see Remark 7, Theorem 1 and Lemma 5) we will prove that $A_h = A_{h,K(h)}$, $W_{h,K(h)} = X_h^*$ (as desired).

In the resulting algorithm, we avoid computing the auxiliary quasi-inverses X_h^*, and all the operations used are similar to ones used for computing A_{h+1} according to (*) provided that Z_h, Y_h and X_h are available cost-free. More precisely, this applies to computation of $A_{h+1,0}$ according to (2). The asymptotic complexity estimates for (2) are

thus the same as for the algorithm of [6] except that the parallel time is now less by the factor of $c\log n$ (for a positive constant c), since here we do not compute X_h^*. Next, for every k, compute $W_{h,k+1}$ and $A_{h,k+1}$ for all h according to (3) and (4). Then again, no quasi-inverses need be computed at these stages, and we only need to perform the matrix additions and multiplications. For every h, we perform $d+\lceil\log_2 n\rceil = O(\log n)$ such matrix operations by using $O(\log^2 n)$ time and

$$O\left(s(n_h)^3/\log n_h\right)$$

processors, under EREW PRAM, and $O(\log n)$ and $s(n_h)^3$ under a randomized CRCW PRAM, where A_h is an $n_h \times n_h$ matrix. We do this concurrently for all $O(\log n)$ values of h, and since n_h decreases as a geometric progression as h grows, we arrive at the overall asymptotic time and processor bounds

$$O(\log^2 n) \quad \text{and} \quad O\left(s(n)^3/\log n\right),$$

respectively, under EREW PRAM, and $O(\log n)$ and $O(s(n)^3)$, under a randomized CRCW PRAM, which improves the asymptotic complexity bounds of [6], as we claimed.

4. The correctness proof

The correctness of the algorithm is immediately implied by Lemma 5, by Remark 6 below and by the following theorem:

Theorem 1. *$A_{h,k} = A_h$, for all h, if $k = h + \lceil\log_2 n\rceil$.*

The theorem follows from the equations (*), (1)–(4), and Lemmas 3 and 5 below (see also Remark 6 below). To prove and even to state Lemmas 3 and 5, we will need some auxiliary definitions.

Definition 2. Hereafter, $(V)_{i,j}$ denotes the (i, j)-entry of a matrix V, $|p|$ denotes the number of distinct edges (that is, the length) of a path p, and $c(p)$ denotes the cost of a path p, defined as the sum of the given costs of all the edges of p. We will assume that all the graphs are complete, for we may interpret any absent edge as an edge of infinite cost.

Lemma 3. *For all h, k, i, and j, we have:*

$$(W_{h,k})_{i,j} \geqslant (X_h^*)_{i,j}, \qquad (A_{h,k})_{i,j} \geqslant (A_h)_{i,j}.$$

Moreover, if $k \geqslant 1$, then

$$(W_{h,k-1})_{i,j} \geqslant (W_{h,k})_{i,j}, \quad (A_{h,k-1})_{i,j} \geqslant (A_{h,k})_{i,j}.$$

Lemma 3 immediately follows from the definition of the recursive factorizations (*) and the equations (1)–(4) (here, we apply induction on k).

Definition 4. We will partition the set V of all the vertices of the graph G associated with the matrix A_0 into the subsets $V_0, V_1, \ldots, V_d$ associated with the matrices $X_0, X_1, \ldots, X_d$ of the recursive factorization (*). (In fact, this partition of the set V is computed in the nested dissection algorithm of [5].) Hereafter, let $\hat{V}_h = \bigcup_{g=0}^{h} V_g$ and let $G_A(\hat{V}_h)$ denote the maximum subgraph of G induced by $\hat{V}_h$ (such that $G_A(\hat{V}_h) = (\hat{V}_h, \hat{E}_h)$, where $\hat{E}_h$ denotes the set of all the edges of G with both endpoints in $\hat{V}_h$).

Lemma 5. *Let p denote a minimum cost path in $G_A(\hat{V}_h)$ between two vertices i and j in $\hat{V}_h$. Let*

$$|p| \leqslant 2^{\max(k-h,0)}. \tag{5}$$

Then

$$(W_{h,k})_{i,j} = (X_h^*)_{i,j} = c(p). \tag{6}$$

Remark 6. The assumption (5) always holds for $k \geqslant h + \lceil\log_2 n\rceil$ and for any minimum cost path p in G, since we suppose that $|p| \leqslant n$ (see the Introduction).

Let $k = d + \lceil\log_2 n\rceil - 1$, so that (5) holds, due to Remark 6. Let us deduce Theorem 1 from Lemmas 3 and 5. Indeed, Lemma 5 implies (6). Furthermore, from Lemma 3, it follows that for all h and k,

$$X_{h,k} \leqslant X_{h,0} = X_h, \qquad Y_{h,k} \leqslant Y_{h,0} = Y_h,$$
$$Z_{h,k} \leqslant Z_{h,0} = Z_h.$$

Substitute these inequalities and (6) into (4) and deduce that

$$A_{h+1,k+1} \leqslant Z_h \oplus Y_h X_h^* Y_h^{\mathrm{T}} = A_{h+1}.$$

Combine the latter inequality with $A_{h+1,k+1} \geqslant A_{h+1}$ due to Lemma 3 and arrive at Theorem 1. □

It remains to prove Lemma 5.

Proof of Lemma 5. Lemma 5 is obvious for $k \leqslant h$, in particular, for $k = 0$. Apply induction on k and always assume that $h < k$. First note that, by the definition of X_h^*, $(X_h^*)_{i,j} = c(p)$, and that, by virtue of Lemma 3, $(W_{h,k})_{i,j} \geqslant (X_h^*)_{i,j}$.

It remains to prove that $(W_{h,k})_{i,j} \leqslant c(p)$. Fix a positive k_0, suppose that the latter unequality holds for all h and for all $k < k_0$, and consider a path p satisfying the assumptions of Lemma 5 for $k = k_0$. Partition p into two subpaths p_1 and p_2 such that $p = p_1 p_2$; the two endpoints of p_s are denoted u_s and v_s for $s = 1, 2$, so that

$$u_1 = i,\ v_1 = u_2,\ v_2 = j \quad \text{where } i, u_2, j \in \hat{V}_h, \quad (7)$$

and

$$|p_s| \leqslant 2^{k_0 - 1 - h}, \quad (8)$$

for $s = 1, 2$.

Note that the induction hypothesis implies that

$$(W_{h,k_0-1})_{u_s,v_s} \leqslant c(p_s), \quad (9)$$

as long as (8) holds.

The inequalities (8) and (9) hold for $s = 1, 2$. On the other hand, (3) implies that

$$W_{h,k_0} = (X_{h,k_0-1} \oplus W_{h,k_0-1})^2,$$

so that

$$(W_{h,k_0})_{i,j} \leqslant c(p_1) + c(p_2) \leqslant c(p).$$

This implies (6). □

This was a graph-theoretical proof. Here is an alternate algebraic proof. Recursively apply (3) and deduce that

$$W_{h,k} \leqslant W_{h,k-1}^2 \leqslant W_{h,0}^{2^k} = (I \oplus X_{h,0})^{2^k} = (I \oplus X_h)^{2^k}$$

for all h and k, but $(I \oplus X_h)^{s(n_h)-1} = X_h^*$, and $s(n_h) \leqslant n_h \leqslant n$. □

Remark 7. The latter proof implies that $W_{h,k} = X_h^*$ and $A_{h,k} = A_h$ already for

$$k = \lceil \log_s(s(n_h) - 1) \rceil \leqslant \lceil \log_2(s(n_0) - 1) \rceil.$$

References

[1] R.C. Backhouse and B.A. Carré, Regular algebra applied to path-finding problems, *J. Inst. Math. Applics.* **15** (1975) 161–186.

[2] B.A. Carré, An algebra for network routing problems, *J. Inst. Math. Applics.* **7** (1971) 273–294.

[3] H. Gazit and J. Reif, A randomized parallel algorithm for planar graph isomorphism, in: *Proc. 2nd Ann. ACM Symp. on Parallel Algorithms and Architecture* (1990) 210–219.

[4] M. Gondran and M. Minoux, *Graphs and Algorithms* (Wiley/Interscience, New York, 1984).

[5] V. Pan and J. Reif, Fast and efficient parallel solution of sparse linear systems, Tech. Rept. 88–19, Computer Science Dept., SUNYA, 1988.

[6] V. Pan and J. Reif, Fast and efficient solution of path algebra problems, *J. Comput. System Sci.* **38** (1989) 494–510.