ISSN 1746-7659, England, UK
Journal of Information and Computing Science
Vol.3, No. 1, 2008, *pp*. 69-72

Permanent: Evaluation by Parallel Algorithm

K. Somasundaram [1+], S. Maria Arulraj [2]

[1] Department of Mathematics, Amrita Vishwa Vidyapeetham, Coimbatore-641 105, India.
[2] Principal, Selvamm Arts and Science College, Namakkal-637 003, India.

(*Received April 13 2007, accepted December 20, 2007*)

Abstract. Permanent of a matrix is # *p*-hard problem shown by many authors. In this paper we present a parallel algorithm for evaluation of permanent of an $n\times n$ matrix with multi-processors.

Keywords: Permanent of matrix, Parallel Algorithm.

1. Introduction

Let $A = (a_{ij})$ be an $m \times n$ matrix, $m \le n$. The permanent of A is defined by $\mathrm{Per}(A) = \sum_{\sigma} \prod_{i=1}^{m} a_{i\,\sigma(i)}$, where the summation extends over all one-to-one functions σ from $(1,2,...,m)$ to $(1,2,...,n)$. The sequence $(a_{1\sigma(1)}, ..., a_{n\sigma(n)})$ is called a diagonal of A and the product $\prod_{i=1}^{m} a_{i\,\sigma(i)}$ is a diagonal product of A. Thus per(A) is the sum of all n! diagonal products of A. The adjacency and incidence matrices of a graph G are denoted by $A(G)$ and $X(G)$ respectively. If $A(G)$ is an adjacency matrix of a bipartite graph G, then the permanent of $A(G)$ is number of perfect matching in G.

The evolution of permanent has attracted the attention of researchers for more than two centuries beginning with Binet and Cauchy in 1812. Despite many attempts, an efficient algorithm for general matrix remains elusive. Ryser's algorithm [11] remains the most efficient for computing the permanent exactly, even through it uses as many as $O(n2^n)$ arithmetic operations. The evaluation of permanent of any matrix of order n using the laplace theorem requires $O((n+1)!)$ multiplications. Many authors [4,5,6,8,12] has developed various methodologies to evaluate the permanent of a matrix with less number of steps using approximate schemes. Valiant [12] has shown that permanent of a non-negative matrix is # *p*-hard, and so it is unlikely to be efficiently computable exactly for all matrices in polynomial time. Planar graphs provides the interesting class of matrices for which a polynomial algorithm of $O(n^3)$ is known [7]. Linial, Samoroditsky and Wigderson [8] have shown a deterministic strongly polynomial algorithm that computes the permanent of a nonnegative matrix of order n within a multiplicative factor of e^n. They developed the first strongly polynomial-time algorithm for matrix scaling - an important nonlinear optimization problem with many applications. Jerrum, Sinclair and Vigoda [5, 6] have given a fully polynomials randomized approximation scheme for computing the permanent of an arbitrary with nonnegative entries using random sampling and Markov chains. Chien, Rasmussen and Sinclair [2] have shown an approximating algorithm for permanent of (0, 1) matrix using Clibord algebra. The basic idea of the paper [2] was to obtain a random matrix B by replacing each 1-entry of A independently by $\pm e$, where e is a random basis element of a suitable algebra; then the output is $|\det(B)|$. Raz [10] has proved that multi-linear formulas for permanent and determinant are of supper-polynomials size.

This paper is organized as follows: Section 2 describes relevant definitions and notations for a tournament graph. Section 3 describes details of the algorithm. In Section 4, the excremental analysis, discussions and conclusions are given.

+ Corresponding author. *E-mail address*: s_sundaram@ettimadai.amrita.edu

2. Path Diagonal Products

Consider a weighted tournament graph D. Let s_i be a directed spanning subgraph of D. The weight $w(s_i)$ of the spanning subgraph s_i is defined to be the product of the weights of the spanning subgraph s_i. The total weight $W(S) = W(S(D)) = \sum_{s_i \in S} \omega(s_i)$, where S is the set of all spanning subgraphs s_i of D. Any weighted tournament D with n vertices can be represented as $n \times n$ matrix A. The rows and columns of A represent the vertices of D, and if the weight of an arc (ij) is a_{ij} in D then the i, j th entry of the matrix A is a_{ij}. In other way, every square matrix A can be represented as a weighted tournament D_A. Each diagonal product in the matrix A gives a spanning subgraph in D_A and similarly each spanning a subgraph in D_A corresponds to a diagonal product in A. The value of diagonal product in A is equal to the product of weights of the corresponding spanning subgraph in D_A, that is $w(s_i)$. Hence sum of all weights of spanning subgraphs in D_A is equal to sum of all diagonal products of the matrix A. Hence per(A) = $W(S(D_A))$.

Example 1: Consider the weight tournament D_A with 3 vertices, the corresponding matrix is

$$A = \begin{pmatrix} 2 & 0 & 2 \\ 1 & 1 & 3 \\ 4 & 0 & 3 \end{pmatrix}$$

$$\text{per}(A) = W(S(D_A)) = \sum_{s_i \in S} \omega(s_i) = 6 + 0 + 0 + 0 + 8 + 0 = 14.$$

Definition 1: Let A be an $n \times n$ matrix. A non vanishing diagonal product in a matrix A is said to be a path diagonal product (*pdp*), if it gives a cycle of length n in the corresponding D_A. Otherwise, the non vanishing diagonal product is called a non path diagonal product (*npdp*).

For a matrix A, the permanent of A is the sum of number of *pdp*'s and number of *npdp*'s of A. In particular for any adjacency or incidence matrix of a graph G, the permanent of A is the sum of number of *pdp*'s and number of *npdp*'s of A. In $A(G)$ each *npdp* is the disconnected spanning subgraphs of G, this is either 2-regular sub graphs or 2- regular sub graphs with at least one single edge or 1- factor directed subgraphs. Consider the graph G with a edge set $E=\{(v_1, v_2), (v_1, v_4), (v_1, v_5), (v_2, v_3), (v_3, v_4), (v_3, v_5), (v_4, v_5)\}$. In $A(G)$, the diagonal product $a_{12}a_{23}a_{35}a_{41}a_{54}$ is a *pdp* where as the diagonal product $a_{12}a_{21}a_{34}a_{45}a_{53}$ is a *npdp*, number of *pdp*'s is 4 and number of *npdp*'s is 4. Hence per$A(G)$ = 4 + 4 = 8.

The number of Hamiltonian cycles in a connected graph G is k if and only if number of *pdp*'s in $A(G)$ is $2k$, since the k Hamiltonian cycles corresponds to $2k$ diagonal products in $A(G)$, which are *pdp*'s in $A(G)$. Also number of Hamiltonian cycles in a connected graph is less than or equal to per$A(G)$.

Lemma 1: If p_1 and p_2 be the number of *npdp*'s in $A(G)$ and $X(G)$ respectively, $p_1 \neq p_2$, then the number of Hamiltonian cycles in G is $k = \frac{1}{2}\left[\frac{p_1\, PerX(G) - p_2\, perA(G)}{perA(G) - PerX(G)}\right]$.

Proof: There are $2k$ *pdp*'s in each of $A(G)$ and $X(G)$, therefore per$A(G) = 2k + p_1$ and Per$X(G) = 2k + p_2$. As $p_1 \neq p_2$, per$A(G) \neq$ Per$X(G)$, $\frac{\text{per}A(G)}{\text{Per}X(G)} = \frac{2k + p_1}{2k + p_2}$ gives the value of k.

Theorem 1: The exact computation of the permanent of an $n \times n$ matrix is # p- complete.

Proof: From the definition, permanent of an $n \times n$ matrix A is sum of *pdp*'s and *npdp*'s. Lemma 1 shows that the number of *pdp*'s is half of number of Hamiltonian cycles in the Tournament D_A, and the computation of k is # p-complete problem. Hence the exact computation of the permanent of an $n \times n$ matrix is # p-complete.

3. Algorithm

In this section, we have shown a parallel algorithm to compute permanent of a matrix of order n. Our algorithm is based on recursive function. Let $A = (a_{ij})$ be a matrix of order n. Let r be the number of processors, then the k^{th} processor, $1 \le k \le r$, will find the permanent for the submatrix corresponding to the non-vanishing diagonal products consisting of a_{1k}, and does the same for the non-vanishing diagonal products consisting of $a_{1(k+ir)}$, $0 \le i \le \left\lfloor \frac{n-k}{r} \right\rfloor$, where $\lfloor x \rfloor$ denotes the greatest integer which is less than or equal to x. That is, the k^{th} processor finds the values of $w(s_{k+ir})$, $0 \le i \le \left\lfloor \frac{n-k}{r} \right\rfloor$. Hence every time, each processor finds the spanning subgraphs, which are all either *pdp*'s or *npdp*'s in D_A. The r^{th} processor finds the total weight $W(S) = \sum_{s_i \in S} \omega(s_i)$, where S is the set of all spanning subgraphs s_i in D_A.

```
Function main ( )
    rank = Processor Id
    r = Number of processors
    n = size of the matrix
If  rank  is root
    Get matrix from the user
    Broadcast the matrix to other processors
    Post receive messages to receive value from all other processors
    For I = rank : n
        Temp = submatrix for the element matrix (1)(I)
        If matrix (1)(I) is not zero
        Value = matrix (1)(I) * sub(temp, size - 1)
        I = I + r
    If  rank  is not root
        Sent the value to root
    If  rank  is root
    Wait until value is received from all the processors
    Add the value received from all the processors.
End Function
Function sub(matrix, size)
    If size is n
        value = permanent of 2 × 2 matrix
    Else
    For I =1 : size
        If matrix (1)(I) is not zero
        value = value + matrix (1)(I) *subfunction(temp, size-1)
End
```

If any one of the values in the diagonal product is zero, then we skip the corresponding diagonal products and find the other non-vanishing diagonal products (which are either *pdp*'s of *npdp*'s). Hence the elimination of zeros in the diagonal products reduces the complexity further. In particular, if the matrix A is a sparse matrix, then the number of computations is very high. The idle time of the r^{th} processor is dependent on the processing times of the other processors, also it is proportional to number of zeros in the sub matrices, the sub matrices are obtained by deleting the first row and $(k + ir)$ columns of the matrix $A = (a_{ij})$, where $0 \le i \le \left\lfloor \frac{n-k}{r} \right\rfloor$. Since the permanent of the matrix A is invariant with respect to the pre/post multiplications of any permutation matrices with A, we can distribute the zeros in A randomly such that the idle time of the r^{th} processor will be negligible.

4. Experimental Results and Conclusion

In a sequentional procedure, the general algorithm for finding the permanent of a matrix runs in a non-polynomial time. In this paper, we use the parallel algorithm for finding permanent of a matrix. The complexity of our algorithm is $\left\lceil \frac{n}{k} \right\rceil$ times the complexity in each processor. We have implemented our algorithm in C on Intel Pentium IV 400 Mhz PCs with 1 GB memory.

Experimental results show that our algorithm is good. We have described a new formulation for evaluating the permanent of a matrix. It is shown that this algorithm is able to minimize its computational complexities of the procedure in evaluating the permanent of a matrix, by implementing in different clusters. Experimental analysis shows that, our implementation is behaving well. Though the permanent has a lot of applications in combinatorial problems and enumeration problems, finding the efficient way to evaluate the permanent is remains elusive. Developing a good sequential procedure for computing the permanent with minimum complexity is still open. Also finding the efficient deterministic polynomial algorithm for approximating permanent is one of the challenging open problems.

5. References

[1] R. A. *Brualdi and H. Ryser*, Combinatorial Matrix Theory,Cambridge University press, 1991.

[2] S.Chien, L Rasmussen and A.Sinclair, Cli_ord Algebras and Approximating the permanent, *STOC*,2002:222-231.

[3] I. T. Foster, *Designing and Building Parallel Programs*, Addison - Wesley Publishing, 1995.

[4] L. M. Goldschlager, *An approximation algorithm for computing the permanent in combinatorial mathematics VII Springer*, Berlin, 1980.

[5] M. Jerum, A. Sinclair and Eric vigoda, A polynomial time approximation algorithm for the permanent of a matrix with non negative entries, *Electronic Colloquium on Computational Complexity*, 2000, **79**.

[6] M.Jerum, A.Sinclar and E. Vigoda, A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries, *Journal of the ACM*, 2004,**51**(4): 671-697.

[7] P. W. Kastelym, The statistics of dimmer as a lattice 1. *The number of dimmer arrangement on a quadrant lattice, Physica*, 1961,**27**:1209–1225.

[8] N. Linial, A. Samoroditsky and A. *Wigderson, A deterministic polynomial algorithm for matrix scalling and approximate permanents*, Proceeding of the 30th Annual ACM symposium on Theory of Computing (STOC) ACM Press, 1998: 644-652.

[9] H. Minc, Permanents, Encyclopedia of Mathematics and its Applications, *Addison-Wesley, Reading, Mass,* 1978, **6**.

[10] R.Raz, Multi-Linear Formulas for Permanent and Determinent are of Supe-Polynomials Size, *STOC*,2004: 633-641.

[11] H. J. Ryser, *Combinatorial Mathematics*, Math. Assoc. Amer. 1963.

[12] L. G. Valiant, The Complexity of Computing the Permanent, *Theoretical Computer Science,* 1979, **8**(2): 189-201.